Contents

Chapter 1

1 to n bit numbers with no consecutive 1s in binary representation

1 to n bit numbers with no consecutive 1s in binary representation - GeeksforGeeks

Given a number n, our task is to find all 1 to n bit numbers with no consecutive 1s in their binary representation.

Examples :-

```
Input : n = 4
Output : 1 2 4 5 8 9 10
These are numbers with 1 to 4
bits and no consecutive ones in
binary representation.

Input : n = 3
Output : 1 2 4 5
```

1) There will be 2^n numbers with number of bits from 1 to n.
2) Iterate through all 2^n numbers. For every number check if it contains consecutive set bits or not. To check, we do bit wise and of current number i and left shifted i. If the bitwise and contains a non-zero bit (or its value is non-zero), then given number doesn't contain consecutive set bits.

C++

```
 // Print all numbers upto n bits
// with no consecutive set bits.
```

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

void printNonConsecutive(int n)
{
    // Let us first compute
    // 2 raised to power n.
    int p = (1 << n);

    // loop 1 to n to check
    // all the numbers
    for (int i = 1; i < p; i++)

        // A number i doesn't contain
        // consecutive set bits if
        // bitwise and of i and left
        // shifted i do't contain a
        // commons set bit.
        if ((i & (i << 1)) == 0)
            cout << i << " ";
}

// Driver code
int main()
{
    int n = 3;
    printNonConsecutive(n);
    return 0;
}
```

Java

```
 // Java Code to Print all numbers upto
// n bits with no consecutive set bits.
import java.util.*;

class GFG
{
    static void printNonConsecutive(int n)
        {
            // Let us first compute
            // 2 raised to power n.
            int p = (1 << n);

            // loop 1 to n to check
            // all the numbers
            for (int i = 1; i < p; i++)
```

```
            // A number i doesn't contain
            // consecutive set bits if
            // bitwise and of i and left
            // shifted i do't contain a
            // commons set bit.
            if ((i & (i << 1)) == 0)
                System.out.print(i + " ");

        }

// Driver code
public static void main(String[] args)
    {
        int n = 3;
        printNonConsecutive(n);
    }
}

// This code is contributed by Mr. Somesh Awasthi
```

Python3

```
 # Python3 program to print all numbers upto
# n bits with no consecutive set bits.

def printNonConsecutive(n):

    # Let us first compute
    # 2 raised to power n.
    p = (1 << n)

    # loop 1 to n to check
    # all the numbers
    for i in range(1, p):

        # A number i doesn't contain
        # consecutive set bits if
        # bitwise and of i and left
        # shifted i do't contain a
        # common set bit.
        if ((i & (i << 1)) == 0):
            print(i, end = " ")

# Driver code
n = 3
printNonConsecutive(n)

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# Code to Print all numbers upto
// n bits with no consecutive set bits.
using System;

class GFG
{
    static void printNonConsecutive(int n)
    {
        // Let us first compute
        // 2 raised to power n.
        int p = (1 << n);

        // loop 1 to n to check
        // all the numbers
        for (int i = 1; i < p; i++)

            // A number i doesn't contain
            // consecutive set bits if
            // bitwise and of i and left
            // shifted i do't contain a
            // commons set bit.
            if ((i & (i << 1)) == 0)
                Console.Write(i + " ");

    }

// Driver code
public static void Main()
    {
        int n = 3;
        printNonConsecutive(n);
    }
}
// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// Print all numbers upto n bits
// with no consecutive set bits.

function printNonConsecutive($n)
{

    // Let us first compute
```

```
    // 2 raised to power n.
    $p = (1 << $n);

    // loop 1 to n to check
    // all the numbers
    for ($i = 1; $i < $p; $i++)

        // A number i doesn't contain
        // consecutive set bits if
        // bitwise and of i and left
        // shifted i do't contain a
        // commons set bit.
        if (($i & ($i << 1)) == 0)
            echo $i . " ";
}

    // Driver code
    $n = 3;
    printNonConsecutive($n);

// This code is contributed by Sam007
?>
```

Complexity O(2^n) because 'for' loop is run 2^n time.

Output:

```
1 2 4 5
```

Improved By : nitin mittal, Sam007

Source

https://www.geeksforgeeks.org/1-n-bit-numbers-no-consecutive-1s-binary-representation/

Chapter 2

1 to n bit numbers with no consecutive 1s in binary representation.

1 to n bit numbers with no consecutive 1s in binary representation. - GeeksforGeeks

Given a number n, our task is to find all 1 to n bit numbers with no consecutive 1s in their binary representation.

Examples:

```
Input : n = 4
Output : 1 2 4 5 8 9 10
These are numbers with 1 to 4
bits and no consecutive ones in
binary representation.

Input : n = 3
Output : 1 2 4 5
```

We add bits one by one and recursively print numbers. For every last bit, we have two choices.

```
   if last digit in sol is 0 then
      we can insert 0 or 1 and recur.
   else if last digit is 1 then
      we can insert 0 only and recur.
```

We will use recursion-

1. We make a solution vector sol and insert first bit 1 in it which will be the first number.
2. Now we check whether length of solution vector is less than or equal to n or not.
3. If it is so then we calculate the decimal number and store it into a map as it store numbers in sorted order.
4. Now we will have two conditions-
    - if last digit in sol is 0 the we can insert 0 or 1 and recur.
    - else if last digit is 1 then we can insert 0 only and recur.

```
numberWithNoConsecutiveOnes(n, sol)
{
if sol.size() <= n

  //  calculate decimal and store it
  if last element of sol is 1
     insert 0 in sol
     numberWithNoConsecutiveOnes(n, sol)
 else
     insert 1 in sol
     numberWithNoConsecutiveOnes(n, sol)

     // because we have to insert zero
     // also in place of 1
     sol.pop_back();
     insert 0 in sol
     numberWithNoConsecutiveOnes(n, sol)
 }


 // CPP program to find all numbers with no
// consecutive 1s in binary representation.
#include <bits/stdc++.h>

using namespace std;
map<int, int> h;

void numberWithNoConsecutiveOnes(int n, vector<int>
                                                sol)
{
    // If it is in limit i.e. of n lengths in
    // binary
    if (sol.size() <= n) {
        int ans = 0;
        for (int i = 0; i < sol.size(); i++)
            ans += pow((double)2, i) *
                   sol[sol.size() - 1 - i];
        h[ans] = 1;
```

```
        // Last element in binary
        int last_element = sol[sol.size() - 1];

        // if element is 1 add 0 after it else
        // If 0 you can add either 0 or 1 after that
        if (last_element == 1) {
            sol.push_back(0);
            numberWithNoConsecutiveOnes(n, sol);
        } else {
            sol.push_back(1);
            numberWithNoConsecutiveOnes(n, sol);
            sol.pop_back();
            sol.push_back(0);
            numberWithNoConsecutiveOnes(n, sol);
        }
    }
}

// Driver program
int main()
{
    int n = 4;
    vector<int> sol;

    // Push first number
    sol.push_back(1);

    // Generate all other numbers
    numberWithNoConsecutiveOnes(n, sol);

    for (map<int, int>::iterator i = h.begin();
                            i != h.end(); i++)
        cout << i->first << " ";
    return 0;
}
```

Output:

```
1 2 4 5 8 9 10
```

Related Post :
Count number of binary strings without consecutive 1's

Improved By : Kishore Srinivas

Source

https://www.geeksforgeeks.org/1-to-n-bit-numbers-with-no-consecutive-1s-in-binary-representation/

Chapter 3

1's and 2's complement of a Binary Number

1's and 2's complement of a Binary Number - GeeksforGeeks

Given a Binary Number as string, print its 1's and 2's complements.

1's complement of a binary number is another binary number obtained by toggling all bits in it, i.e., transforming the 0 bit to 1 and the 1 bit to 0.

Examples:

```
1's complement of "0111" is "1000"
1's complement of "1100" is  "0011"
```

2's complement of a binary number is 1 added to the 1's complement of the binary number. Examples:

```
2's complement of "0111" is  "1001"
2's complement of "1100" is  "0100"
```

For one's complement, we simply need to flip all bits.

For 2's complement, we first find one's complement. We traverse the one's complement starting from LSB (least significant bit), and look for 0. We flip all 1's (change to 0) until we find a 0. Finally, we flip the found 0. For example, 2's complement of "01000" is "11000" (Note that we first find one's complement of 01000 as 10111). If there are all 1's (in one's complement), we add an extra 1 in the string. For example, 2's complement of "000" is "1000" (1's complement of "000" is "111").

Below is C++ implementation.

```
 // C++ program to print 1's and 2's complement of
// a binary number
#include <bits/stdc++.h>
using namespace std;

// Returns '0' for '1' and '1' for '0'
char flip(char c) {return (c == '0')? '1': '0';}

// Print 1's and 2's complement of binary number
// represented by "bin"
void printOneAndTwosComplement(string bin)
{
    int n = bin.length();
    int i;

    string ones, twos;
    ones = twos = "";

    //  for ones complement flip every bit
    for (i = 0; i < n; i++)
        ones += flip(bin[i]);

    //  for two's complement go from right to left in
    //  ones complement and if we get 1 make, we make
    //  them 0 and keep going left when we get first
    //  0, make that 1 and go out of loop
    twos = ones;
    for (i = n - 1; i >= 0; i--)
    {
        if (ones[i] == '1')
            twos[i] = '0';
        else
        {
            twos[i] = '1';
            break;
        }
    }

    // If No break : all are 1  as in 111  or  11111;
    // in such case, add extra 1 at beginning
    if (i == -1)
        twos = '1' + twos;


    cout << "1's complement: " << ones << endl;
    cout << "2's complement: " << twos << endl;
}
```

```
// Driver program
int main()
{
    string bin = "1100";
    printOneAndTwosComplement(bin);
    return 0;
}
```

Output:

```
1's complement: 0011
2's complement: 0100
```

Thanks to Utkarsh Trivedi for above solution.

As a side note, signed numbers generally use 2's complement representation. Positive values are stored as it is and negative values are stored in their 2's complement form. One extra bit is required to indicate whether number is positive or negative. For example char is 8 bits in C. If 2's complement representation is used for char, then 127 is stored as it is, i.e., 01111111 where first 0 indicates positive. But -127 is stored as 10000001.

Related Post :
Efficient method for 2's complement of a binary string

References:
http://qa.geeksforgeeks.org/6439/write-program-calculate-ones-and-twos-complement-of-number
http://geeksquiz.com/whats-difference-between-1s-complement-and-2s-complement/

Source

https://www.geeksforgeeks.org/1s-2s-complement-binary-number/

Chapter 4

A Boolean Array Puzzle

A Boolean Array Puzzle - GeeksforGeeks

Input: A array arr[] of two elements having value 0 and 1

Output: Make both elements 0.

Specifications: Following are the specifications to follow.
1) It is guaranteed that one element is 0 but we do not know its position.
2) We can't say about another element it can be 0 or 1.
3) We can only complement array elements, no other operation like and, or, multi, division, ... etc.
4) We can't use if, else and loop constructs.
5) Obviously, we can't directly assign 0 to array elements.

There are several ways we can do it as we are sure that always one Zero is there. Thanks to devendraiiit for suggesting following 3 methods.

Method 1

```
 void changeToZero(int a[2])
{
   a[ a[1] ] = a[ !a[1] ];
}

int main()
{
   int a[] = {1, 0};
   changeToZero(a);

   printf(" arr[0] = %d \n", a[0]);
   printf(" arr[1] = %d ", a[1]);
   getchar();
   return 0;
}
```

Method 2

```
 void changeToZero(int a[2])
{
    a[ !a[0] ] = a[ !a[1] ]
}
```

Method 3
This method doesn't even need complement.

```
 void changeToZero(int a[2])
{
    a[ a[1] ] = a[ a[0] ]
}
```

Method 4
Thanks to **purvi** for suggesting this method.

```
 void changeToZero(int a[2])
{
  a[0] = a[a[0]];
  a[1] = a[0];
}
```

There may be many more methods.

Source

https://www.geeksforgeeks.org/a-boolean-array-puzzle/

Chapter 5

A backtracking approach to generate n bit Gray Codes

A backtracking approach to generate n bit Gray Codes - GeeksforGeeks

Given a number n, the task is to generate n bit Gray codes (generate bit patterns from 0 to 2^n-1 such that successive patterns differ by one bit)
Examples:

```
Input : 2
Output : 0 1 3 2
Explanation :
00 - 0
01 - 1
11 - 3
10 - 2

Input : 3
Output : 0 1 3 2 6 7 5 4
```

We have discussed an approach in Generate n-bit Gray Codes

This article provides a **backtracking approach** to the same problem. Idea is that for each bit out of n bit we have a choice either we can ignore it or we can invert the bit so this means our gray sequence goes upto 2 ^ n for n bits. So we make two recursive calls for either inverting the bit or leaving the bit as it is.

```
 // CPP program to find the gray sequence of n bits.
#include <iostream>
#include <vector>
using namespace std;
```

```
/* we have 2 choices for each of the n bits either we
   can include i.e invert the bit or we can exclude the
   bit i.e we can leave the number as it is. */
void grayCodeUtil(vector<int>& res, int n, int& num)
{
    // base case when we run out bits to process
    // we simply include it in gray code sequence.
    if (n == 0) {
        res.push_back(num);
        return;
    }

    // ignore the bit.
    grayCodeUtil(res, n - 1, num);

    // invert the bit.
    num = num ^ (1 << (n - 1));
    grayCodeUtil(res, n - 1, num);
}

// returns the vector containing the gray
// code sequence of n bits.
vector<int> grayCodes(int n)
{
    vector<int> res;

    // num is passed by reference to keep
    // track of current code.
    int num = 0;
    grayCodeUtil(res, n, num);

    return res;
}

// Driver function.
int main()
{
    int n = 3;
    vector<int> code = grayCodes(n);
    for (int i = 0; i < code.size(); i++)
        cout << code[i] << endl;
    return 0;
}
```

Output:

```
0
1
3
2
6
7
5
4
```

Source

https://www.geeksforgeeks.org/backtracking-approach-generate-n-bit-gray-codes/

Chapter 6

Add 1 to a given number

Add 1 to a given number - GeeksforGeeks

Write a program to add one to a given number. The use of operators like '+', '-', '*', '/', '++', '–' ...etc are not allowed.
Examples:

```
Input:  12
Output: 13

Input:  6
Output: 7
```

This question can be approached by using some bit magic. Following are different methods to achieve the same using bitwise operators.

Method 1
To add 1 to a number x (say 0011000111), flip all the bits after the rightmost 0 bit (we get 0011000000). Finally, flip the rightmost 0 bit also (we get 0011001000) to get the answer.

C

```
 // C++ code to add add
// one to a given number
#include <stdio.h>

int addOne(int x)
{
    int m = 1;

    // Flip all the set bits
    // until we find a 0
```

```
    while( x & m )
    {
        x = x ^ m;
        m <<= 1;
    }

    // flip the rightmost 0 bit
    x = x ^ m;
    return x;
}

/* Driver program to test above functions*/
int main()
{
    printf("%d", addOne(13));
    getchar();
    return 0;
}
```

Java

```
 // Java code to add add
// one to a given number
class GFG {

    static int addOne(int x)
    {
        int m = 1;

        // Flip all the set bits
        // until we find a 0
        while( (int)(x & m) == 1)
        {
            x = x ^ m;
            m <<= 1;
        }

        // flip the rightmost 0 bit
        x = x ^ m;
        return x;
    }

    /* Driver program to test above functions*/
    public static void main(String[] args)
    {
        System.out.println(addOne(13));
    }
}
```

```
// This code is contributed by prerna saini.
```

Python3

```
 # Python3 code to add 1
# one to a given number
def addOne(x) :

    m = 1;
    # Flip all the set bits
    # until we find a 0
    while(x & m):
        x = x ^ m
        m <<= 1

    # flip the rightmost
    # 0 bit
    x = x ^ m
    return x

# Driver program
n = 13
print addOne(n)

# This code is contributed by Prerna Saini.
```

C#

```
 // C# code to add one
// to a given number
using System;

class GFG {

    static int addOne(int x)
    {
        int m = 1;

        // Flip all the set bits
        // until we find a 0
        while( (int)(x & m) == 1)
        {
            x = x ^ m;
            m <<= 1;
        }
```

```
        // flip the rightmost 0 bit
        x = x ^ m;
        return x;
    }

    // Driver code
    public static void Main()
    {
        Console.WriteLine(addOne(13));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP code to add add
// one to a given number


function addOne($x)
{
    $m = 1;

    // Flip all the set bits
    // until we find a 0
    while( $x & $m )
    {
        $x = $x ^ $m;
        $m <<= 1;
    }

    // flip the rightmost 0 bit
    $x = $x ^ $m;
    return $x;
}

// Driver Code
echo addOne(13);

// This code is contributed by vt_m.
?>
```

Output:

```
14
```

Method 2
We know that the negative number is represented in 2's complement form on most of the architectures. We have the following lemma hold for 2's complement representation of signed numbers.

Say, x is numerical value of a number, then

~x = -(x+1) [~ is for bitwise complement]

(x + 1) is due to addition of 1 in 2's complement conversion

To get (x + 1) apply negation once again. So, the final expression becomes (-(~x)).

C

```
 #include<stdio.h>

int addOne(int x)
{
   return (-(~x));
}

/* Driver program to test above functions*/
int main()
{
  printf("%d", addOne(13));
  getchar();
  return 0;
}
```

Java

```
 // Java code to Add 1 to a given number
class GFG
{
    static int addOne(int x)
    {
         return (-(~x));
    }

    // Driver program
    public static void main(String[] args)
    {
        System.out.printf("%d", addOne(13));
    }
}

// This code is contributed
// by Smitha Dinesh Semwal
```

Python3

```
 # Python3 code to add 1 to a given number

def addOne(x):
    return (-(~x));


# Driver program
print(addOne(13))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# code to Add 1
// to a given number
using System;

class GFG
{
    static int addOne(int x)
    {
        return (-(~x));
    }

    // Driver program
    public static void Main()
    {
        Console.WriteLine(addOne(13));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Code to Add 1
// to a given number

function addOne($x)
{
return (-(~$x));
}

// Driver Code
```

```
echo addOne(13);

// This code is contributed by vt_m.
?>
```

Output:

```
14
```

Example

```
Assume the machine word length is one *nibble* for simplicity.
And x = 2 (0010),
~x = ~2 = 1101 (13 numerical)
-~x = -1101
```

Interpreting bits 1101 in 2's complement form yields numerical value as -(2^4 – 13) = -3. Applying '-' on the result leaves 3. Same analogy holds for decrement. Note that this method works only if the numbers are stored in 2's complement form.

Thanks to *Venki* for suggesting this method.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/add-1-to-a-given-number/

Chapter 7

Add minimum number to an array so that the sum becomes even

Add minimum number to an array so that the sum becomes even - GeeksforGeeks

Given an array, write a program to add the minimum number(**should be greater than 0**) to the array so that the sum of array becomes even.

Examples:

```
Input : 1 2 3 4 5 6 7 8
Output : 2
Explanation : Sum of array is 36, so we
add minimum number 2 to make the sum even.

Input : 1 2 3 4 5 6 7 8 9
Output : 1
```

Method 1 (Computing Sum). We calculate the sum of all elements of the array, then we can check if the sum is even minimum number is 2, else minimum number is 1. This method can cause overflow if sum exceeds allowed limit.

Method 2. Instead of calculating the sum of numbers, we keep the **count of odd number of elements in the array**. If count of odd numbers present is even we return 2, else we return 1.
For example – Array contains : 1 2 3 4 5 6 7
Odd number counts is 4. And we know that the **sum of even numbers of odd number is even**. And sum of even number is always even (that is why, we don't keep count of even numbers).

C++

```
 // CPP program to add minimum number
// so that the sum of array becomes even
#include <iostream>
using namespace std;

// Function to find out minimum number
int minNum(int arr[], int n)
{
    // Count odd number of terms in array
    int odd = 0;
    for (int i = 0; i < n; i++)
        if (arr[i] % 2)
            odd += 1;

   return (odd % 2)? 1 : 2;
}

// Driver code
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << minNum(arr, n) << "n";

    return 0;
}
```

Java

```
 // Java program to add minimum number
// so that the sum of array becomes even

class GFG
{
    // Function to find out minimum number
    static int minNum(int arr[], int n)
    {
        // Count odd number of terms in array
        int odd = 0;
        for (int i = 0; i < n; i++)
            if (arr[i] % 2 != 0)
                odd += 1;

        return ((odd % 2) != 0)? 1 : 2;
```

```
    }

    // Driver method to test above function
    public static void main(String args[])
    {
        int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        int n = arr.length;

        System.out.println(minNum(arr, n));
    }
}
```

Python

```
 # Python program to add minimum number
# so that the sum of array becomes even

# Function to find out minimum number
def minNum(arr, n):

    # Count odd number of terms in array
    odd = 0
    for i in range(n):
        if (arr[i] % 2):
            odd += 1

    if (odd % 2):
        return 1
    return 2


# Driver code
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
n = len(arr)
print minNum(arr, n)
```

C#

```
 // C# program to add minimum number
// so that the sum of array becomes even
using System;

class GFG
{
    // Function to find out minimum number
    static int minNum(int []arr, int n)
    {
```

```
        // Count odd number of terms in array
        int odd = 0;
        for (int i = 0; i < n; i++)
            if (arr[i] % 2 != 0)
                odd += 1;

        return ((odd % 2) != 0)? 1 : 2;
    }

    // Driver Code
    public static void Main()
    {
        int []arr = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        int n = arr.Length;

        Console.Write(minNum(arr, n));
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP program to add minimum number
// so that the sum of array becomes even

// Function to find out minimum number
function minNum( $arr, $n)
{

    // Count odd number of
    // terms in array
    $odd = 0;
    for ($i = 0; $i < $n; $i++)
        if ($arr[$i] % 2)
            $odd += 1;

    return ($odd % 2)? 1 : 2;
}

    // Driver code
    $arr = array(1, 2, 3, 4, 5,
                 6, 7, 8, 9);
    $n = count($arr);
    echo minNum($arr, $n) ;

// This code is contributed by anuj_67.
```

```
?>
```

Output:

```
1
```

Method 3. We can also improve the 2 method, we don't need to keep count of number of odd elements present. We can **take a boolean variable** (initialized as 0). Whenever we **find the odd element in the array we perform the NOT(!) operation** on the boolean variable. This logical operator inverts the value of the boolean variable (meaning if it is 0, it converts the variable to 1 and vice-versa).
For example – Array contains : 1 2 3 4 5
Explanation : variable initialized as 0.
Traversing the array
1 is odd, applying NOT operation in variable, now variable becomes 1.
2 is even, no operation.
3 is odd, applying NOT operation in variable, now variable becomes 0.
4 is even, no operation.
5 is odd, applying NOT operation in variable, now variable becomes 1.

If **variable value is 1 it means odd number of odd elements are present**, minimum number to make sum of elements even is by adding 1.
Else minimum number is 2.

C++

```
 // CPP program to add minimum number
// so that the sum of array becomes even

#include <iostream>
using namespace std;

// Function to find out minimum number
int minNum(int arr[], int n)
{
    // Count odd number of terms in array
    bool odd = 0;
    for (int i = 0; i < n; i++)
        if (arr[i] % 2)
            odd = !odd;

    if (odd)
        return 1;
    return 2;
}

// Driver code
```

```
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << minNum(arr, n) << "n";

    return 0;
}
```

Java

```
 // Java program to add minimum number
// so that the sum of array becomes even

class GFG
{
    // Function to find out minimum number
    static int minNum(int arr[], int n)
    {
        // Count odd number of terms in array
        Boolean odd = false;
        for (int i = 0; i < n; i++)
            if (arr[i] % 2 != 0)
                odd = !odd;

        if (odd)
            return 1;
        return 2;
    }

    //Driver method to test above function
    public static void main(String args[])
    {
        int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        int n = arr.length;

        System.out.println(minNum(arr, n));
    }
}
```

Python

```
 # Python program to add minimum number
# so that the sum of array becomes even

# Function to find out minimum number
```

```
def minNum(arr, n):

    # Count odd number of terms in array
    odd = False
    for i in range(n):
        if (arr[i] % 2):
            odd = not odd
    if (odd):
        return 1
    return 2


# Driver code
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
n = len(arr)
print minNum(arr, n)
```

C#

```
 // C# program to add minimum number
// so that the sum of array becomes even
using System;

class GFG
{

    // Function to find out minimum number
    static int minNum(int []arr, int n)
    {
        // Count odd number of terms in array
        bool odd = false;
        for (int i = 0; i < n; i++)
            if (arr[i] % 2 != 0)
                odd = !odd;

        if (odd)
            return 1;
        return 2;
    }

    //Driver Code
    public static void Main()
    {
        int []arr = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        int n = arr.Length;

        Console.Write(minNum(arr, n));
    }
```

```
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP program to add minimum number
// so that the sum of array becomes even

// Function to find out minimum number
function minNum($arr, $n)
{

    // Count odd number of
    // terms in array
    $odd = 0;
    for($i = 0; $i < $n; $i++)
        if ($arr[$i] % 2)
            $odd = !$odd;

    if ($odd)
        return 1;
    return 2;
}

    // Driver code
    $arr = array(1, 2, 3, 4, 5, 6, 7, 8, 9);
    $n = sizeof($arr);
    echo minNum($arr, $n) ,"\n";

// This code is contributed by nitin mittal
?>
```

Output :

```
1
```

Exercise :
Find the minimum number required to make the sum of elements odd.

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/add-minimum-number-to-an-array-so-that-the-sum-becomes-even/

Chapter 8

Add two bit strings

Add two bit strings - GeeksforGeeks

Given two bit sequences as strings, write a function to return the addition of the two sequences. Bit strings can be of different lengths also. For example, if string 1 is “1100011” and second string 2 is “10”, then the function should return “1100101”.

Since sizes of two strings may be different, we first make the size of smaller string equal to that of bigger string by adding leading 0s. After making sizes same, we one by one add bits from rightmost bit to leftmost bit. In every iteration, we need to sum 3 bits: 2 bits of 2 given strings and carry. The sum bit will be 1 if, either all of the 3 bits are set or one of them is set. So we can do XOR of all bits to find the sum bit. How to find carry – carry will be 1 if any of the two bits is set. So we can find carry by taking OR of all pairs. Following is step by step algorithm.

1. Make them equal sized by adding 0s at the begining of smaller string.
2. Perform bit addition
.....Boolean expression for adding 3 bits a, b, c
.....Sum = a XOR b XOR c
.....Carry = (a AND b) OR (b AND c) OR (c AND a)

Following is C++ implementation of the above algorithm.

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

//adds the two bit strings and return the result
string addBitStrings( string first, string second );

// Helper method: given two unequal sized bit strings, converts them to
// same length by aadding leading 0s in the smaller string. Returns the
// the new length
int makeEqualLength(string &str1, string &str2)
{
```

```
    int len1 = str1.size();
    int len2 = str2.size();
    if (len1 < len2)
    {
        for (int i = 0 ; i < len2 - len1 ; i++)
            str1 = '0' + str1;
        return len2;
    }
    else if (len1 > len2)
    {
        for (int i = 0 ; i < len1 - len2 ; i++)
            str2 = '0' + str2;
    }
    return len1; // If len1 >= len2
}

// The main function that adds two bit sequences and returns the addition
string addBitStrings( string first, string second )
{
    string result;  // To store the sum bits

    // make the lengths same before adding
    int length = makeEqualLength(first, second);

    int carry = 0;  // Initialize carry

    // Add all bits one by one
    for (int i = length-1 ; i >= 0 ; i--)
    {
        int firstBit = first.at(i) - '0';
        int secondBit = second.at(i) - '0';

        // boolean expression for sum of 3 bits
        int sum = (firstBit ^ secondBit ^ carry)+'0';

        result = (char)sum + result;

        // boolean expression for 3-bit addition
        carry = (firstBit & secondBit) | (secondBit & carry) | (firstBit & carry);
    }

    // if overflow, then add a leading 1
    if (carry)
        result = '1' + result;

    return result;
}
```

```
// Driver program to test above functions
int main()
{
    string str1 = "1100011";
    string str2 = "10";

    cout << "Sum is " << addBitStrings(str1, str2);
    return 0;
}
```

Output:

```
Sum is 1100101
```

This article is compiled by **Ravi Chandra Enaganti**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/add-two-bit-strings/

Chapter 9

Add two numbers without using arithmetic operators

Add two numbers without using arithmetic operators - GeeksforGeeks

Write a function Add() that returns sum of two integers. The function should not use any of the arithmetic operators (+, ++, –, -, .. etc).

Sum of two bits can be obtained by performing XOR (^) of the two bits. Carry bit can be obtained by performing AND (&) of two bits.
Above is simple Half Adder logic that can be used to add 2 single bits. We can extend this logic for integers. If x and y don't have set bits at same position(s), then bitwise XOR (^) of x and y gives the sum of x and y. To incorporate common set bits also, bitwise AND (&) is used. Bitwise AND of x and y gives all carry bits. We calculate (x & y) << 1 and add it to x ^ y to get the required result.

C

```
 // C Program to add two numbers
// without using arithmetic operator
#include<stdio.h>

int Add(int x, int y)
{
    // Iterate till there is no carry
    while (y != 0)
    {
        // carry now contains common
        //set bits of x and y
        int carry = x & y;

        // Sum of bits of x and y where at
        //least one of the bits is not set
```

```
        x = x ^ y;

        // Carry is shifted by one so that adding
        // it to x gives the required sum
        y = carry << 1;
    }
    return x;
}

int main()
{
    printf("%d", Add(15, 32));
    return 0;
}
```

Java

```
 // Java Program to add two numbers
// without using arithmetic operator
import java.io.*;

class GFG
{
    static int Add(int x, int y)
    {
        // Iterate till there is no carry
        while (y != 0)
        {
            // carry now contains common
            // set bits of x and y
            int carry = x & y;

            // Sum of bits of x and
            // y where at least one
            // of the bits is not set
            x = x ^ y;

            // Carry is shifted by
            // one so that adding it
            // to x gives the required sum
            y = carry << 1;
        }
        return x;
    }

    // Driver code
    public static void main(String arg[])
    {
```

```
        System.out.println(Add(15, 32));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 Program to add two numbers
# without using arithmetic operator
def Add(x, y):

    # Iterate till there is no carry
    while (y != 0):

        # carry now contains common
        # set bits of x and y
        carry = x & y

        # Sum of bits of x and y where at
        # least one of the bits is not set
        x = x ^ y

        # Carry is shifted by one so that
        # adding it to x gives the required sum
        y = carry << 1

    return x

print(Add(15, 32))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# Program to add two numbers
// without using arithmetic operator
using System;

class GFG
{
    static int Add(int x, int y)
    {
        // Iterate till there is no carry
        while (y != 0)
        {
```

```
            // carry now contains common
            // set bits of x and y
            int carry = x & y;

            // Sum of bits of x and
            // y where at least one
            // of the bits is not set
            x = x ^ y;

            // Carry is shifted by
            // one so that adding it
            // to x gives the required sum
            y = carry << 1;
        }
        return x;
    }

    // Driver code
    public static void Main()
    {
        Console.WriteLine(Add(15, 32));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to add two numbers
// without using arithmetic operator

function Add( $x, $y)
{

    // Iterate till there is
    // no carry
    while ($y != 0)
    {

        // carry now contains common
        //set bits of x and y
        $carry = $x & $y;

        // Sum of bits of x and y where at
        //least one of the bits is not set
        $x = $x ^ $y;
```

```
        // Carry is shifted by one
        // so that adding it to x
        // gives the required sum
        $y = $carry << 1;
    }
    return $x;
}

    // Driver Code
    echo Add(15, 32);

// This code is contributed by anuj_67.
?>
```

Output :

```
47
```

Following is the recursive implementation for the same approach.

```
 int Add(int x, int y)
{
    if (y == 0)
        return x;
    else
        return Add( x ^ y, (x & y) << 1);
}
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/add-two-numbers-without-using-arithmetic-operators/

Chapter 10

Addition of two numbers without carry

Addition of two numbers without carry - GeeksforGeeks

You are given two positive number n and m. You have to find simply addition of both number but with a given condition that there is not any carry system in this addition. That is no carry is added at higher MSBs.

Examples :

```
Input : m = 456, n = 854
Output : 200

Input : m = 456, n = 4
Output : 450
```

Algorithm :

Input n, m

while(n||m) { // Add each bits bit_sum = (n%10) + (m%10); // Neglect carry bit_sum %= 10; // Update result // multiplier to maintain place value res = (bit_sum * multiplier) + res; n /= 10; m /= 10; // Update multiplier multiplier *=10; }

print res

Approach :
To solve this problem we will need the bit by bit addition of number where we start adding two number from right most bit (LSB) and add integers from both nubers with same position.

Also we will neglect carry at each position so that that carry will not affect further higher bit position.
Start adding both numbers bit by bit and for each bit take sum of integers then neglect their carry by taking modulo of bit_sum by 10 further add bit_sum to res by multiplying bit_sum with a multiplier specifying place value. (Multiplier got incremented 10 times on each iteration.)

Below is the implementation of above approach :

C++

```
 // CPP program for special
// addition of two number
#include <bits/stdc++.h>
using namespace std;

int xSum(int n, int m)
{
    // variable to store result
    int res = 0;

    // variable to maintain
    // place value
    int multiplier = 1;

    // variable to maintain
    // each digit sum
    int bit_sum;

    // Add numbers till each
    // number become zero
    while (n || m) {

        // Add each bits
        bit_sum = (n % 10) + (m % 10);

        // Neglect carry
        bit_sum %= 10;

        // Update result
        res = (bit_sum * multiplier) + res;
        n /= 10;
        m /= 10;

        // Update multiplier
        multiplier *= 10;
    }
```

```
    return res;
}

// Driver program
int main()
{
    int n = 8458;
    int m = 8732;
    cout << xSum(n, m);
    return 0;
}
```

Java

```
 // Java program for special
// addition of two number
import java.util.*;
import java.lang.*;

public class GfG {

    public static int xSum(int n, int m)
    {
        int res = 0;
        int multiplier = 1;
        int bit_sum;

        // Add numbers till each
        // number become zero
        while (true) {

            // Add each bits
            bit_sum = (n % 10) + (m % 10);

            // Neglect carry
            bit_sum %= 10;

            // Update result
            res = (bit_sum * multiplier) + res;
            n /= 10;
            m /= 10;

            // Update multiplier
            multiplier *= 10;
            if (n == 0)
                break;
            if (m == 0)
                break;
```

```
        }
        return res;
    }

    // Driver function
    public static void main(String args[])
    {
        int n = 8458;
        int m = 8732;
        System.out.println(xSum(n, m));
    }
}
/* This code is contributed by Sagar Shukla */
```

Python3

```
 # Python3 program for special
# addition of two number
import math

def xSum(n, m) :

    # variable to
    # store result
    res = 0

    # variable to maintain
    # place value
    multiplier = 1

    # variable to maintain
    # each digit sum
    bit_sum = 0

    # Add numbers till each
    # number become zero
    while (n or m) :

        # Add each bits
        bit_sum = ((n % 10) +
                   (m % 10))

        # Neglect carry
        bit_sum = bit_sum % 10

        # Update result
        res = (bit_sum *
               multiplier) + res
```

```
        n = math.floor(n / 10)
        m = math.floor(m / 10)

        # Update multiplier
        multiplier = multiplier * 10

    return res

# Driver code
n = 8458
m = 8732
print (xSum(n, m))

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# program for special
// addition of two number
using System;

public class GfG {

    public static int xSum(int n, int m)
    {
        int res = 0;
        int multiplier = 1;
        int bit_sum;

        // Add numbers till each
        // number become zero
        while (true) {

            // Add each bits
            bit_sum = (n % 10) + (m % 10);

            // Neglect carry
            bit_sum %= 10;

            // Update result
            res = (bit_sum * multiplier) + res;
            n /= 10;
            m /= 10;

            // Update multiplier
            multiplier *= 10;
            if (n == 0)
```

```
                break;
            if (m == 0)
                break;
        }
        return res;
    }

    // Driver function
    public static void Main()
    {
        int n = 8458;
        int m = 8732;
        Console.WriteLine(xSum(n, m));
    }
}

/* This code is contributed by Vt_m */
```

PHP

```
 <?php
// php program for special
// addition of two number

function xSum($n, $m)
{

    // variable to store result
    $res = 0;

    // variable to maintain
    // place value
    $multiplier = 1;

    // variable to maintain
    // each digit sum
    $bit_sum;

    // Add numbers till each
    // number become zero
    while ($n || $m) {

        // Add each bits
        $bit_sum = ($n % 10) +
                   ($m % 10);

        // Neglect carry
        $bit_sum %= 10;
```

```
        // Update result
        $res = ($bit_sum * $multiplier) + $res;
        $n =floor($n / 10);
        $m =floor($m / 10);

        // Update multiplier
        $multiplier *= 10;
    }
    return $res;
}

    // Driver code
    $n = 8458;
    $m = 8732;
    echo xSum($n, $m);

//This code is contributed by mits
?>
```

Output :

```
6180
```

Improved By : Mithun Kumar, manishshaw1

Source

https://www.geeksforgeeks.org/special-addition-two-number/

Chapter 11

Alternate bits of two numbers to create a new number

Alternate bits of two numbers to create a new number - GeeksforGeeks

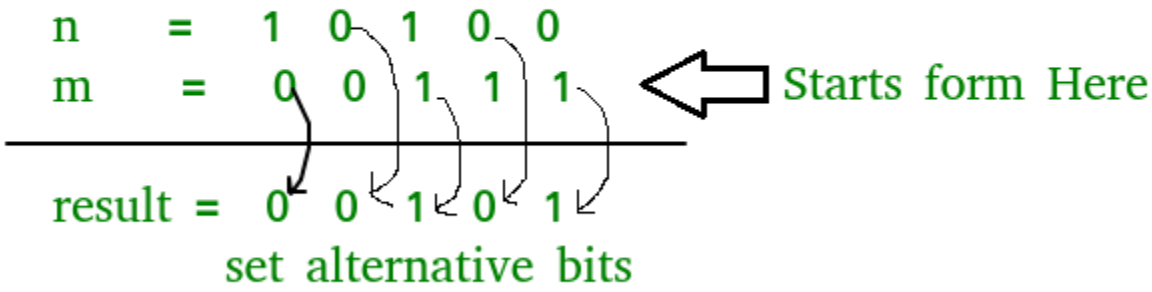
Given two numbers, the task is to use alternative bits within two numbers to create result. We take first bits of second number, then second bit of the first number, third bit of second number and take the fourth bit of a first number and so on and generate a number with it.

Examples :

```
Input : n = 10, m = 11
Output : 11
Start from right of second number
Binary representation of n = 1 0 1 0
                               ^   ^
Binary representation of m = 1 0 1 1
                                 ^   ^
Output  is                 = 1 0 1 1

Input : n = 20, m = 7
Output : 5
Start from right of second number
binary representation of n = 1 0 1 0 0
                             ^   ^
binary representation of m = 0 0 1 1 1
                           ^   ^   ^
Output  is                 = 0 0 1 0 1
```

Approach :-
1. Get the set even bits number of n.

2. Get the set odd bits number of m.
3. return OR of these number.



C++

```
 // CPP Program to generate a number using
// alternate bits of two numbers.
#include <iostream>
using namespace std;

// set even bit of number n
int setevenbits(int n)
{
    int temp = n;
    int count = 0;

    // res for store 101010.. number
    int res = 0;

    // generate number form of 101010.....
    // till temp size
    for (temp = n; temp > 0; temp >>= 1) {

        // if bit is even then generate
        // number and or with res
        if (count % 2 == 1)
            res |= (1 << count);

        count++;
    }
```

```
    // return set even bit number
    return (n & res);
}

// set odd bit of number m
int setoddbits(int m)
{
    int count = 0;

    // res for store 101010.. number
    int res = 0;

    // generate number form of 101010....
    // till temp size
    for (int temp = m; temp > 0; temp >>= 1) {

        // if bit is even then generate
        // number and or with res
        if (count % 2 == 0)
            res |= (1 << count);

        count++;
    }

    // return set odd bit number
    return (m & res);
}

int getAlternateBits(int n, int m)
{
    // set even bit of number n
    int tempn = setevenbits(n);

    // set odd bit of number m
    int tempm = setoddbits(m);

    // take OR with these number
    return (tempn | tempm);
}

// Driver code
int main()
{
    int n = 10;
    int m = 11;

    // n  = 1 0 1 0
```

```
    //        ^   ^
    // m  = 1 0 1 1
    //          ^   ^
    // result= 1 0 1 1

    cout << getAlternateBits(n, m);

    return 0;
}
```

Java

```
 // java Program to generate a number using
// alternate bits of two numbers.
import java.io.*;

class GFG {

    // set even bit of number n
    static int setevenbits(int n)
    {

        int temp = n;
        int count = 0;

        // res for store 101010.. number
        int res = 0;

        // generate number form of 101010.....
        // till temp size
        for (temp = n; temp > 0; temp >>= 1) {

            // if bit is even then generate
            // number and or with res
            if (count % 2 == 1)
                res |= (1 << count);

            count++;
        }

        // return set even bit number
        return (n & res);
    }

    // set odd bit of number m
    static int setoddbits(int m)
    {
        int count = 0;
```

```
        // res for store 101010.. number
        int res = 0;

        // generate number form of 101010....
        // till temp size
        for (int temp = m; temp > 0; temp >>= 1)
        {
            // if bit is even then generate
            // number and or with res
            if (count % 2 == 0)
                res |= (1 << count);

            count++;
        }

        // return set odd bit number
        return (m & res);
    }

    static int getAlternateBits(int n, int m)
    {
        // set even bit of number n
        int tempn = setevenbits(n);

        // set odd bit of number m
        int tempm = setoddbits(m);

        // take OR with these number
        return (tempn | tempm);
    }

    // Driver code
    public static void main (String[] args)
    {

    int n = 10;
    int m = 11;

    // n = 1 0 1 0
    //       ^   ^
    // m = 1 0 1 1
    //         ^   ^
    // result= 1 0 1 1
    System.out.println(getAlternateBits(n, m));
    }
}
```

```
// This code is contributed by vt_m
```

Python3

```
 # Python Program to generate a number using
# alternate bits of two numbers.

# set even bit of number n
def setevenbits(n):
    temp = n
    count = 0

    # res for store 101010.. number
    res = 0

    # generate number form of 101010.....
    # till temp size
    while temp > 0:

        # if bit is even then generate
        # number and or with res
        if count % 2:
            res |= (1 << count)

        count += 1
        temp >>= 1

    # return set even bit number
    return (n & res)

# set odd bit of number m
def setoddbits(m):
    temp = m
    count = 0

    # res for store 101010.. number
    res = 0

    # generate number form of 101010....
    # till temp size
    while temp > 0:

        # if bit is even then generate
        # number and or with res
        if not count % 2:
            res |= (1 << count)

        count += 1
```

```
        temp >>= 1

    # return set odd bit number
    return (m & res)

def getAlternateBits(n, m):
    # set even bit of number n
    tempn = setevenbits(n)

    # set odd bit of number m
    tempm = setoddbits(m)

    # take OR with these number
    return (tempn | tempm)

# Driver code
n = 10
m = 11

# n = 1 0 1 0
#     ^ ^
# m = 1 0 1 1
#         ^ ^
# result= 1 0 1 1

print(getAlternateBits(n, m))

# This code is contributed by Ansu Kumari.
```

C#

```
 // C# Program to generate a number using
// alternate bits of two numbers.
using System;

class GFG {

    // set even bit of number n
    static int setevenbits(int n)
    {

        int temp = n;
        int count = 0;

        // res for store 101010.. number
        int res = 0;

        // generate number form of 101010.....
```

```
    // till temp size
    for (temp = n; temp > 0; temp >>= 1) {

        // if bit is even then generate
        // number and or with res
        if (count % 2 == 1)
            res |= (1 << count);

        count++;
    }

    // return set even bit number
    return (n & res);
}

// set odd bit of number m
static int setoddbits(int m)
{
    int count = 0;

    // res for store 101010.. number
    int res = 0;

    // generate number form of 101010....
    // till temp size
    for (int temp = m; temp > 0; temp >>= 1)
    {
        // if bit is even then generate
        // number and or with res
        if (count % 2 == 0)
            res |= (1 << count);

        count++;
    }

    // return set odd bit number
    return (m & res);
}

static int getAlternateBits(int n, int m)
{
    // set even bit of number n
    int tempn = setevenbits(n);

    // set odd bit of number m
    int tempm = setoddbits(m);

    // take OR with these number
```

```
        return (tempn | tempm);
    }

    // Driver code
    public static void Main ()
    {

        int n = 10;
        int m = 11;

        // n = 1 0 1 0
        // ^ ^
        // m = 1 0 1 1
        //     ^ ^
        // result= 1 0 1 1
        Console.WriteLine(getAlternateBits(n, m));
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP Program to generate a number using
// alternate bits of two numbers.

// set even bit of number n
function setevenbits($n)
{
    $temp = $n;
    $count = 0;

    // res for store 101010.. number
    $res = 0;

    // generate number form of 101010.....
    // till temp size
    for ($temp = $n; $temp > 0; $temp >>= 1)
    {

        // if bit is even then generate
        // number and or with res
        if ($count % 2 == 1)
            $res |= (1 << $count);

        $count++;
    }
```

```
    // return set even bit number
    return ($n & $res);
}

// set odd bit of number m
function setoddbits($m)
{
    $count = 0;

    // res for store 101010.. number
    $res = 0;

    // generate number form of 101010....
    // till temp size
    for ($temp = $m; $temp > 0; $temp >>= 1)
    {

        // if bit is even then generate
        // number and or with res
        if ($count % 2 == 0)
            $res |= (1 << $count);

        $count++;
    }

    // return set odd bit number
    return ($m & $res);
}

function getAlternateBits($n, $m)
{
    // set even bit of number n
    $tempn = setevenbits($n);

    // set odd bit of number m
    $tempm = setoddbits($m);

    // take OR with these number
    return ($tempn | $tempm);
}

// Driver code
$n = 10;
$m = 11;

// n = 1 0 1 0
// ^ ^
```

```
// m = 1 0 1 1
//     ^ ^
// result= 1 0 1 1

echo getAlternateBits($n, $m);

// This code is contributed by mits
?>
```

Output :

```
11
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/alternate-bits-of-two-numbers-to-create-a-new-number/

Chapter 12

Baum Sweet Sequence

Baum Sweet Sequence - GeeksforGeeks

Baum Sweet Sequence is an infinite binary sequence of 0s and 1s. The nth term of the sequence is 1 if the number n has an odd number of contiguous zeroes in its binary representation, else the nth term is 0.

```
The first few terms of the sequence are:
b1 = 1 (binary of 1 is 1)
b2 = 0 (binary of 2 is 10)
b3 = 1 (binary of 3 is 11)
b4 = 1 (binary of 4 is 100)
b5 = 0 (binary of 5 is 101)
b6 = 0 (binary of 6 is 110)
```

Given a natural number **n**. The task is to find the nth term of the Baum Sweet sequence, i.e, check whether it contains any consecutive block of zeroes of odd length.

```
Input: n = 8
Output: 0
Explanations:
Binary representation of 8 is 1000. It
contains odd length block of consecutive 0s.
Therefore B8 is 0.

Input: n = 5
Output: 1

Input: n = 7
Output: 0
```

The idea is to run a loop through the binary representation of n and count the length of all the consecutive zero blocks present. If there is at-least one odd length zero block, then the nth term for the given input n is 0 else it is 1.

```
 // CPP code to find the nth term of the
// Baum Sweet Sequence
#include <bits/stdc++.h>
using namespace std;

int nthBaumSweetSeq(int n)
{
    // bitset stores bitwise representation
    bitset<32> bs(n);

    // len stores the number of bits in the
    // binary of n. builtin_clz() function gives
    // number of zeroes present before the
    // leading 1 in binary of n
    int len = 32 - __builtin_clz(n);

    int baum = 1; // nth term of baum sequence
    for (int i = 0; i < len;) {
        int j = i + 1;

        // enter into a zero block
        if (bs[i] == 0) {
            int cnt = 1;

            // loop to run through each zero block
            // in binary representation of n
            for (j = i + 1; j < len; j++) {

                // counts consecutive zeroes
                if (bs[j] == 0)
                    cnt++;
                else
                    break;
            }

            // check if the number of consecutive
            // zeroes is odd
            if (cnt % 2 == 1)
                baum = 0;
        }
        i = j;
    }

    return baum;
```

```
}

// Driver Code
int main()
{
    int n = 8;
    cout << nthBaumSweetSeq(n);
    return 0;
}
```

Output:

```
0
```

Source

https://www.geeksforgeeks.org/baum-sweet-sequence/

Chapter 13

Binary representation of a given number

Binary representation of a given number - GeeksforGeeks

Write a program to print Binary representation of a given number.

Source: Microsoft Interview Set-3

Method 1: Iterative
For any number, we can check whether its 'i'th bit is 0(OFF) or 1(ON) by bitwise ANDing it with "2^i" (2 raise to i).

```
1) Let us take number 'NUM' and we want to check whether it's 0th bit is ON or OFF
    bit = 2 ^ 0 (0th bit)
    if  NUM & bit == 1 means 0th bit is ON else 0th bit is OFF

2) Similarly if we want to check whether 5th bit is ON or OFF
    bit = 2 ^ 5 (5th bit)
    if NUM & bit == 1 means its 5th bit is ON else 5th bit is OFF.
```

Let us take unsigned integer (32 bit), which consist of 0-31 bits. To print binary representation of unsigned integer, start from 31th bit, check whether 31th bit is ON or OFF, if it is ON print "1" else print "0". Now check whether 30th bit is ON or OFF, if it is ON print "1" else print "0", do this for all bits from 31 to 0, finally we will get binary representation of number.

```
 void bin(unsigned n)
{
    unsigned i;
    for (i = 1 << 31; i > 0; i = i / 2)
        (n & i)? printf("1"): printf("0");
```

```
}

int main(void)
{
    bin(7);
    printf("\n");
    bin(4);
}
```

Method 2: Recursive
Following is recursive method to print binary representation of 'NUM'.

```
step 1) if NUM > 1
    a) push NUM on stack
    b) recursively call function with 'NUM / 2'
step 2)
    a) pop NUM from stack, divide it by 2 and print it's remainder.

 void bin(unsigned n)
{
    /* step 1 */
    if (n > 1)
        bin(n/2);

    /* step 2 */
    printf("%d", n % 2);
}

int main(void)
{
    bin(7);
    printf("\n");
    bin(4);
}
```

Method 3: Recursive using bitwise operator
Steps to convert decimal number to its binary representation are given below:

```
step 1: Check n > 0
step 2: Right shift the number by 1 bit and recursive function call
step 3: Print the bits of number

 #include<bits/stdc++.h>
using namespace std;
```

```
// Function to convert decimal
// to binary number
void bin(unsigned n)
{
    if (n > 1)
    bin(n>>1);

    printf("%d", n & 1);
}

// Driver code
int main(void)
{
    bin(131);
    printf("\n");
    bin(3);
    return 0;
}
```

This article is compiled by **Narendra Kangralkar**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : TheNaiveKid

Source

https://www.geeksforgeeks.org/binary-representation-of-a-given-number/

Chapter 14

Binary representation of previous number

Binary representation of previous number - GeeksforGeeks

Given a binary input that represents binary representation of positive number n, find binary representation of n-1. It may be assumed that input binary number is greater than 0.

The binary input may or may not fit even in unsigned long long int.

Examples:

```
Input : 10110
Output : 10101
Here n  = (22)10 = (10110)2
Previous number = (21)10 = (10101)2

Input : 11000011111000000
Output : 11000011110111111
```

We store input as string so that large numbers can be handled. We traverse the string from rightmost character and convert all 0's to 1's until we find a 1. Finally convert the found 1 to 0. The number so formed after this process is the required number. If input is "1", then previous number will be "0". If only the first character in the entire string is '1', then we discard this character and change all the 0's to 1's.

```
 // C++ implementation to find the binary
// representation of previous number
#include <bits/stdc++.h>
using namespace std;

// function to find the required
```

```
// binary representation
string previousNumber(string num)
{
    int n = num.size();

    // if the number is '1'
    if (num.compare("1") == 0)
        return "0";

    // examine bits from right to left
    int i;
    for (i = n - 1; i >= 0; i--) {

        // if '1' is encountered, convert
        // it to '0' and then break
        if (num.at(i) == '1') {
            num.at(i) = '0';
            break;
        }

        // else convert '0' to '1'
        else
            num.at(i) = '1';
    }

    // if only the 1st bit in the
    // binary representation was '1'
    if (i == 0)
        return num.substr(1, n - 1);

    // final binary representation
    // of the required number
    return num;
}

// Driver program to test above
int main()
{
    string num = "10110";
    cout << "Binary representation of previous number = "
         << previousNumber(num);
    return 0;
}
```

Output:

```
Binary representation of previous number = 10101
```

Time Complexity : O(n) where **n** is number of bits in input.

Source

https://www.geeksforgeeks.org/binary-representation-previous-number/

Chapter 15

Bit Fields in C

Bit Fields in C - GeeksforGeeks

In C, we can specify size (in bits) of structure and union members. The idea is to use memory efficiently when we know that the value of a field or group of fields will never exceed a limit or is withing a small range.

For example, consider the following declaration of date without use of bit fields.

```
 #include <stdio.h>

// A simple representation of date
struct date
{
   unsigned int d;
   unsigned int m;
   unsigned int y;
};

int main()
{
   printf("Size of date is %d bytes\n", sizeof(struct date));
   struct date dt = {31, 12, 2014};
   printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
}
```

Output:

```
Size of date is 12 bytes
Date is 31/12/2014
```

The above representation of 'date' takes 12 bytes on a compiler where an unsigned int takes 4 bytes. Since we know that the value of d is always from 1 to 31, value of m is from 1 to 12, we can optimize the space using bit fields.

```
 #include <stdio.h>

// A space optimized representation of date
struct date
{
   // d has value between 1 and 31, so 5 bits
   // are sufficient
   unsigned int d: 5;

   // m has value between 1 and 12, so 4 bits
   // are sufficient
   unsigned int m: 4;

   unsigned int y;
};

int main()
{
   printf("Size of date is %d bytes\n", sizeof(struct date));
   struct date dt = {31, 12, 2014};
   printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
   return 0;
}
```

Output:

```
Size of date is 8 bytes
Date is 31/12/2014
```

Following are some interesting facts about bit fields in C.

1) A special unnamed bit field of size 0 is used to force alignment on next boundary. For example consider the following program.

```
 #include <stdio.h>

// A structure without forced alignment
struct test1
{
   unsigned int x: 5;
   unsigned int y: 8;
};

// A structure with forced alignment
struct test2
{
   unsigned int x: 5;
   unsigned int: 0;
```

```
   unsigned int y: 8;
};

int main()
{
   printf("Size of test1 is %d bytes\n", sizeof(struct test1));
   printf("Size of test2 is %d bytes\n", sizeof(struct test2));
   return 0;
}
```

Output:

```
Size of test1 is 4 bytes
Size of test2 is 8 bytes
```

2) We cannot have pointers to bit field members as they may not start at a byte boundary.

```
 #include <stdio.h>
struct test
{
   unsigned int x: 5;
   unsigned int y: 5;
   unsigned int z;
};
int main()
{
   struct test t;

   // Uncommenting the following line will make
   // the program compile and run
   printf("Address of t.x is %p", &t.x);

   // The below line works fine as z is not a
   // bit field member
   printf("Address of t.z is %p", &t.z);
   return 0;
}
```

Output:

```
 error: attempt to take address of bit-field structure member 'test::x'
```

3) It is implementation defined to assign an out-of-range value to a bit field member.

```
 #include <stdio.h>
struct test
{
   unsigned int x: 2;
   unsigned int y: 2;
   unsigned int z: 2;
};
int main()
{
   struct test t;
   t.x = 5;
   printf("%d", t.x);
   return 0;
}
```

Output:

```
Implementation-Dependent
```

4) In C++, we can have static members in a structure/class, but bit fields cannot be static.

```
 // The below C++ program compiles and runs fine
struct test1 {
   static unsigned int x;
};
int main() {  }


// But below C++ program fails in compilation as bit fields
// cannot be static
struct test1 {
   static unsigned int x: 5;
};
int main() {  }
// error: static member 'x' cannot be a bit-field
```

5) Array of bit fields is not allowed. For example, the below program fails in compilation.

```
 struct test
{
  unsigned int x[10]: 5;
};
```

```
int main()
{

}
```

Output:

```
error: bit-field 'x' has invalid type
```

Exercise:
Predict the output of following programs. Assume that unsigned int takes 4 bytes and long int takes 8 bytes.
1)

```
 #include <stdio.h>
struct test
{
   unsigned int x;
   unsigned int y: 33;
   unsigned int z;
};
int main()
{
   printf("%d", sizeof(struct test));
   return 0;
}
```

2)

```
 #include <stdio.h>
struct test
{
   unsigned int x;
   long int y: 33;
   unsigned int z;
};
int main()
{
   struct test t;
   unsigned int *ptr1 = &t.x;
   unsigned int *ptr2 = &t.z;
   printf("%d", ptr2 - ptr1);
   return 0;
}
```

3)

```
 union test
{
  unsigned int x: 3;
  unsigned int y: 3;
  int z;
};

int main()
{
   union test t;
   t.x = 5;
   t.y = 4;
   t.z = 1;
   printf("t.x = %d, t.y = %d, t.z = %d",
           t.x, t.y, t.z);
   return 0;
}
```

4) Use bit fields in C to figure out a way whether a machine is little endian or big endian.

Source

https://www.geeksforgeeks.org/bit-fields-c/

Chapter 16

Bit Tricks for Competitive Programming

Bit Tricks for Competitive Programming - GeeksforGeeks

In competitive programming or in general some problems seems difficult but can be solved very easily with little bit magic. We have discussed some tricks in below previous post.

Bitwise Hacks for Competitive Programming

We have considered below facts in this article –

- 0 based indexing of bits from left to right.
- Setting i-th bit means, turning i-th bit to 1
- Clearing i-th bit means, turning i-th bit to 0

1) Clear all bits from LSB to ith bit

```
mask = ~((1 << i+1 ) - 1);
x &= mask;
```

Logic: To clear all bits from LSB to i-th bit, we have to AND x with mask having LSB to i-th bit 0. To obtain such mask, first left shift 1 i times. Now if we minus 1 from that, all the bits from 0 to i-1 become 1 and remaining bits become 0. Now we can simply take complement of mask to get all first i bits to 0 and remaining to 1.
Example-
x = 29 (00011101) and we want to clear LSB to 3rd bit, total 4 bits
mask -> 1 << 4 -> 16(00010000)
mask -> 16 – 1 -> 15(00001111)
mask -> ~mask -> 11110000
x & mask -> 16 (00010000)

2) Clearing all bits from MSB to i-th bit

```
mask = (1 << i) - 1;
x &= mask;
```

Logic: To clear all bits from MSB to i-th bit, we have to AND x with mask having MSB to i-th bit 0. To obtain such mask, first left shift 1 i times. Now if we minus 1 from that, all the bits from 0 to i-1 become 1 and remaining bits become 0.
Example-
x = 215 (11010111) and we want to clear MSB to 4th bit, total 4 bits
mask -> 1 << 4 -> 16(00010000)
mask -> 16 – 1 -> 15(00001111)
x & mask -> 7(00000111)

3) Divide by 2

```
x >>= 1;
```

Logic: When we do arithmetic right shift, every bit is shifted to right and blank position is substituted with sign bit of number, 0 in case of positive and 1 in case of negative number. Since every bit is a power of 2, with each shift we are reducing the value of each bit by factor of 2 which is equivalent to division of x by 2.
Example-
x = 18(00010010)
x >> 1 = 9 (00001001)

4) Multiplying by 2

```
x <<= 1;
```

Logic: When we do arithmetic left shift, every bit is shifted to left and blank position is substituted with 0 . Since every bit is a power of 2, with each shift we are increasing the value of each bit by a factor of 2 which is equivalent to multiplication of x by 2.
Example-
x = 18(00010010)
x << 1 = 36 (00100100)

5) Upper case English alphabet to lower case

```
ch |= ' ';
```

Logic: The bit representation of upper case and lower case English alphabets are –

```
A -> 01000001          a -> 01100001
```

```
B -> 01000010          b -> 01100010
C -> 01000011          c -> 01100011
  .                         .
  .                         .
Z -> 01011010          z -> 01111010
```

As we can see if we set 5th bit of upper case characters, it will be converted into lower case character. We have to prepare a mask having 5th bit 1 and other 0 (00100000). This mask is bit representation of space character (‘ ‘). The character ‘ch’ then ORed with mask.

Example-
ch = ‘A’ (01000001)
mask = ‘ ‘ (00100000)
ch | mask = ‘a’ (01100001)
Please refer Case conversion (Lower to Upper and Vice Versa) for details.

6) Lower case English alphabet to upper case

```
ch &= '_' ;
```

Logic: The bit representation of upper case and lower case English alphabets are –

```
A -> 01000001              a -> 01100001
B -> 01000010              b -> 01100010
C -> 01000011              c -> 01100011
.                             .
.                             .
Z -> 01011010              z -> 01111010
```

As we can see if we clear 5th bit of lower case characters, it will be converted into upper case character. We have to prepare a mask having 5th bit 0 and other 1 (10111111). This mask is bit representation of underscore character (‘_‘). The character ‘ch’ then AND with mask.
Example-
ch = ‘a’ (01100001)
mask = ‘_ ‘ (11011111)
ch | mask = ‘A’ (01000001)
Please refer Case conversion (Lower to Upper and Vice Versa) for details.

7) Count set bits in integer

```
 int countSetBits(int x)
{
    int count = 0;
    while (x)
    {
```

```
        x &= (x-1);
        count++;
    }
    return count;
}
```

Logic: This is**Brian Kernighan's** algorithm.

8) Find log base 2 of 32 bit integer

```
 int log2(int x)
{
    int res = 0;
    while (x >>= 1)
        res++;
    return res;
}
```

Logic: We right shift x repeatedly until it becomes 0, meanwhile we keep count on the shift operation. This count value is the log2(x).

9) Checking if given 32 bit integer is power of 2

```
 int isPowerof2(int x)
{
    return (x && !(x & x-1));
}
```

Logic: All the power of 2 have only single bit set e.g. 16 (00010000). If we minus 1 from this, all the bits from LSB to set bit get toggled, i.e., 16-1 = 15 (00001111). Now if we AND x with (x-1) and the result is 0 then we can say that x is power of 2 otherwise not. We have to take extra care when x = 0.

Example
x = 16(000100000)
x – 1 = 15(00001111)
x & (x-1) = 0
so 16 is power of 2

Please refer this article for more bit hacks.

Source

https://www.geeksforgeeks.org/bit-tricks-competitive-programming/

Chapter 17

Bitmasking and Dynamic Programming | Set-2 (TSP)

Bitmasking and Dynamic Programming | Set-2 (TSP) - GeeksforGeeks

In this post, we will be using our knowledge of dynamic programming and Bitmasking technique to solve one of the famous NP-hard problem "Travelling Salesman Problem".

Before solving the problem, we assume that the reader has the knowledge of

- DP and formation of DP transition relation
- Bitmasking in DP
- Travelling Salesman problem

To understand this concept lets consider the below problem :

Problem Description:

```
Given a 2D grid of characters representing
a town where '*' represents the
houses, '#' represents the blockage,
'.' represents the vacant street
area. Currently you are (0, 0) position.

Our task is to determine the minimum distance
to be moved to visit all the houses and return
to our initial position at (0, 0). You can
only move to adjacent cells that share exactly
1 edge with the current cell.
```

The above problem is the well-known Travelling Salesman Problem.

The first part is to calculate the minimum distance between the two cells. We can do it by simply using a BFS as all the distances are unit distance. To optimize our solution we will be pre-calculating the distances taking the initial location and the location of the houses as the source point for our BFS.

Each BFS traversal takes O(size of grid) time. Therefore, it is **O(X * size_of_grid)** for overall pre-calculation, where X = number of houses + 1 (initial position)

Now lets's think of a DP state

So we will be needing to track the visited houses and the last visited house to uniquely identify a state in this problem.

Therefore, we will be taking **dp[index][mask]** as our DP state.

Here,
index : tells us the location of current house
mask : tells us the houses that are visited (if ith bit is set in mask then this means that the ith dirty tile is cleaned)

Whereas dp[index][mask] will tell us the minimum distance to visit X(number of set bits in mask) houses corresponding to their order of their occurrence in the mask where the last visited house is house at location index.

State transition relation

So our initial state will be **dp[0][0]** this tells that we are currently at initial tile that is our initial location and mask is 0 that states that no house is visited till now.

And our final destination state will be **dp[any index][LIMIT_MASK]**, here LIMIT_MASK = **(1<<N) − 1**
and N = number of houses.

Therefore our DP state transition can be stated as

```
dp(curr_idx)(curr_mask) = min{
    for idx : off_bits_in_curr_mask
       dp(idx)(cur_mask.set_bit(idx)) + dist[curr_idx][idx]
}
```

The above relation can be visualized as the minimum distance to visit all the houses by standing at curr_idx house and by already visiting cur_mask houses is equal to min of distance between the curr_idx house and idx house + minimum distance to visit all the houses by standing at idx house and by already
visiting (**cur_mask | (1 <<idx)**) houses.

So, here we iterate over all possible idx values such that cur_mask has i^{th} bit as 0 that tells us that i^{th} house is not visited.

Whenever we have our mask = LIMIT_MASK, this means that we have visited all the houses in the town. So, we will add the distance from the last visited town (i.e the town at cur_idx positon) to the initial position (0, 0).

The C++ program for the above implementation is given below:

```
 #include <bits/stdc++.h>
using namespace std;

#define INF 99999999
#define MAXR 12
#define MAXC 12
#define MAXMASK 2048
#define MAXHOUSE 12

// stores distance taking souce
// as every dirty tile
int dist[MAXR][MAXC][MAXHOUSE];

// memoization for dp states
int dp[MAXHOUSE][MAXMASK];

// stores coordinates for
// dirty tiles
vector < pair < int, int > > dirty;

// Directions
int X[] = {-1, 0, 0, 1};
int Y[] = {0, 1, -1, 0};

char arr[21][21];

// len : number of dirty tiles + 1
// limit : 2 ^ len -1
// r, c : number of rows and columns
int len, limit, r, c;


// Returns true if current position
// is safe to visit
// else returns false
// Time Complexity : O(1)
bool safe(int x, int y)
{
    if (x >= r or y>= c or x<0 or y<0)
       return false;
    if (arr[x][y] == '#')
       return false;
    return true;
}


// runs BFS traversal at tile idx
// calulates distance to every cell
```

```
// in the grid
// Time Complexity : O(r*c)
void getDist(int idx){

    // visited array to track visited cells
    bool vis[21][21];
    memset(vis, false, sizeof(vis));

    // getting current positon
    int cx = dirty[idx].first;
    int cy = dirty[idx].second;

    // initializing queue for bfs
    queue < pair < int, int > > pq;
    pq.push({cx, cy});

    // initializing the dist to max
    // because some cells cannot be visited
    // by taking source cell as idx
    for (int i = 0;i<= r;i++)
        for (int j = 0;j<= c;j++)
            dist[i][j][idx] = INF;

    // base conditions
    vis[cx][cy] = true;
    dist[cx][cy][idx] = 0;

    while (! pq.empty())
    {
        auto x = pq.front();
        pq.pop();
        for (int i = 0;i<4;i++)
        {
           cx = x.first + X[i];
           cy = x.second + Y[i];
           if (safe(cx, cy))
           {
               if (vis[cx][cy])
                   continue;
               vis[cx][cy] = true;
               dist[cx][cy][idx] = dist[x.first][x.second][idx] + 1;
               pq.push({cx, cy});
            }
         }
    }
}

// Dynamic Programming state transition recursion
```

```
// with memoization. Time Complexity: O(n*n*2 ^ n)
int solve(int idx, int mask)
{
    // goal state
    if (mask == limit)
       return dist[0][0][idx];

    // if already visited state
    if (dp[idx][mask] != -1)
       return dp[idx][mask];

    int ret = INT_MAX;

    // state transiton relation
    for (int i = 0;i<len;i++)
    {
        if ((mask & (1<<i)) == 0)
        {
            int newMask = mask | (1<<i);
            ret = min( ret, solve(i, newMask)
                + dist[dirty[i].first][dirty[i].second][idx]);
        }
    }

    // adding memoization and returning
    return dp[idx][mask] = ret;
}

void init()
{
    // initializing containers
    memset(dp, -1, sizeof(dp));
    dirty.clear();

    // populating dirty tile positions
    for (int i = 0;i<r;i++)
        for (int j = 0;j<c;j++)
        {
            if (arr[i][j] == '*')
               dirty.push_back({i, j});
        }

    // inserting ronot's location at the
    // begining of the dirty tile
    dirty.insert(dirty.begin(), {0, 0});

    len = dirty.size();
```

```
    // calculating LIMIT_MASK
    limit = (1<<len) - 1;

    // precalculating distances from all
    // dirty tiles to each cell in the grid
    for (int i = 0;i<len;i++)
       getDist(i);
}

int main(int argc, char const *argv[])
{
    // Test case #1:
    //     .....*.
    //     ...#...
    //     .*.#.*.
    //     .......

char A[4][7] = {    {'.', '.', '.', '.', '.', '*', '.'},
                    {'.', '.', '.', '#', '.', '.', '.'},
                    {'.', '*', '.', '#', '.', '*', '.'},
                    {'.', '.', '.', '.', '.', '.', '.'}
                };

    r = 4; c = 7;

    cout << "The given grid : " << endl;

    for (int i = 0;i<r;i++)
    {
        for (int j = 0;j<c;j++)
        {
            cout << A[i][j] << " ";
            arr[i][j] = A[i][j];
        }
        cout << endl;
    }

    // - initializitiation
    // - precalculations
    init();

    int ans = solve(0, 1);

    cout << "Minimum distance for the given grid : ";
    cout << ans << endl;


    // Test Case #2
```

```
    //     ...#...
    //     ...#.*.
    //     ...#...
    //     .*.#.*.
    //     ...#...

    char Arr[5][7] = {  {'.', '.', '.', '#', '.', '.', '.'},
                        {'.', '.', '.', '#', '.', '*', '.'},
                        {'.', '.', '.', '#', '.', '.', '.'},
                        {'.', '*', '.', '#', '.', '*', '.'},
                        {'.', '.', '.', '#', '.', '.', '.'}
                };

    r = 5; c = 7;

    cout << "The given grid : " << endl;

    for (int i = 0;i<r;i++)
    {
        for (int j = 0;j<c;j++)
        {
            cout << Arr[i][j] << " ";
            arr[i][j] = Arr[i][j];
        }
        cout << endl;
    }

    // - initializitiation
    // - precalculations
    init();
    ans = solve(0, 1);
    cout << "Minimum distance for the given grid : ";
    if (ans >= INF)
        cout << "not possible" << endl;
    else
        cout << ans << endl;

    return 0;
}
```

Output:

```
The given grid :
. . . . . * .
. . . # . . .
. * . # . * .
. . . . . . .
```

```
Minimum distance for the given grid : 16
The given grid :
. . . # . . .
. . . # . * .
. . . # . . .
. * . # . * .
. . . # . . .
Minimum distance for the given grid : not possible
```

Note:

We have used the initial state to be **dp[0][1]** because we have pushed the start location at the first position in the container of houses. Hence, our Bit Mask will be 1 as the 0^{th} bit is set i.e we have visited the starting location for our trip.

Time Complexity:

Consider the number of houses to be **n**. So, there are **n * (2^n)** states and at every state, we are looping over n houses to transit over to next state and because of memoization we are doing this looping transition only once for each state. Therefore, our Time Complexity is **O(n^2 * 2^n)**.

Recommended:

- http://www.spoj.com/problems/CLEANRBT/
- https://www.youtube.com/watch?v=-JjA4BLQyqE

Source

https://www.geeksforgeeks.org/bitmasking-dynamic-programming-set-2-tsp/

Chapter 18

Bits manipulation (Important tactics)

Bits manipulation (Important tactics) - GeeksforGeeks

Prerequisites : Bitwise operators in C, Bitwise Hacks for Competitive Programming, Bit Tricks for Competitive Programming

1. Compute XOR from 1 to n (direct method) :

```
 // Direct XOR of all numbers from 1 to n
int computeXOR(int n)
{
    if (n % 4 == 0)
        return n;
    if (n % 4 == 1)
        return 1;
    if (n % 4 == 2)
        return n + 1;
    else
        return 0;
}
```

```
Input: 6
Output: 7
```

   Refer Compute XOR from 1 to n for details.

2. We can quickly calculate the total number of combinations with numbers smaller than or equal to with a number whose sum and XOR are equal. Instead of using looping (Brute force method), we can directly find it by a mathematical trick i.e.

```
// Refer Equal Sum and XOR for details.
Answer = pow(2, count of zero bits)
```

3. How to know if a number is a power of 2?

```
 //  Function to check if x is power of 2
bool isPowerOfTwo(int x)
{
     // First x in the below expression is
     // for  the case when x is 0
     return x && (!(x & (x - 1)));
}
```

   Refer check if a number is power of twofor details.

4. Find XOR of all subsets of a set. We can do it in O(1) time. The answer is always 0 if given set has more than one elements. For set with single element, the answer is value of single element. Refer XOR of the XOR's of all subsets for details.

5. We can quickly find number of leading, trailing zeroes and number of 1's in a binary code of an integer in C++ using GCC. It can be done by using inbuilt function i.e.

```
Number of leading zeroes: builtin_clz(x)
Number of trailing zeroes : builtin_ctz(x)
Number of 1-bits: __builtin_popcount(x)
```

   ReferGCC inbuilt functions for details.

6. Convert binary code directly into an integer in C++.

```
 // Conversion into Binary code//
#include <iostream>
using namespace std;

int main()
{
    auto number = 0b011;
    cout << number;
    return 0;
}
```

```
Output: 3
```

7. The Quickest way to swap two numbers:

```
a ^= b;
b ^= a;
a ^= b;
```

Refer swap two numbersfor details.

8. Simple approach to flip the bits of a number: It can be done by a simple way, just simply subtract the number from the value obtained when all the bits are equal to 1 . For example:

```
Number : Given Number
Value  : A number with all bits set in given number.
Flipped number = Value - Number.

Example :
Number = 23,
Binary form: 10111;
After flipping digits number will be: 01000;
Value: 11111 = 31;
```

9. We can find the most significant set bit in O(1) time for a fixed size integer. For example below cocde is for 32 bit integer.

```
 int setBitNumber(int n)
{
    // Below steps set bits after
    // MSB (including MSB)

    // Suppose n is 273 (binary
    // is 100010001). It does following
    // 100010001 | 010001000 = 110011001
    n |= n>>1;

    // This makes sure 4 bits
    // (From MSB and including MSB)
    // are set. It does following
    // 110011001 | 001100110 = 111111111
    n |= n>>2;

    n |= n>>4;
    n |= n>>8;
    n |= n>>16;

    // Increment n by 1 so that
    // there is only one set bit
    // which is just before original
    // MSB. n now becomes 1000000000
```

```
    n = n + 1;

    // Return original MSB after shifting.
    // n now becomes 100000000
    return (n >> 1);
}
```

Refer Find most significant set bit of a number for details.

10. We can quickly check if bits in a number are in alternate pattern (like 101010). We compute n ^ (n >> 1). If n has an alternate pattern, then n ^ (n >> 1) operation will produce a number having set bits only. '^' is a bitwise XOR operation. Refer check if a number has bits in alternate pattern for details.

Source

https://www.geeksforgeeks.org/bits-manipulation-important-tactics/

Chapter 19

Bitwise OR (or |) of a range

Bitwise OR (or |) of a range - GeeksforGeeks

Given two integers L and R. Determine the bitwise OR of all the integers in the range [L, R] (both inclusive).

Examples:

```
Input: L = 3, R = 8
Output: 15
3 | 4 | 5 | 6 | 7 | 8 = 15

Input: L = 12, R = 18
Output: 31
12 | 13 | 14 | 15 | 16 | 17 | 18 = 31
```

A **naive** approach is to traverse through all the integers between L and R and do bitwise OR of all the numbers.

An **efficient** approach be to follow the following steps:

1. Find the position of Most Significant Bit (MSB) in both the numbers (L and R)
2. If the position of both MSBs are different, set all the bits from the max(MSB1, MSB2) including this different bit upto Oth bit i.e. add the value (1 << i) for all 0 i max(MSB1, MSB2) in the answer.
3. If the position of both MSBs are same, then
    - Set this bit corresponding to MSB or add the value (1 << MSB) in the answer.
    - Subtract the value (1 << MSB) from both the numbers (L and R).
    - Repeat the steps 1, 2 and 3.

Given below is the working of the above algorithm when L = 18 and R = 21.

```
L = 18, R = 21
The result is initially 0.
The position of Most Significant Bit in L = 4
Position of Most Significant Bit in R = 4
Since positions are same, add value (1 << 4) i.e. 16 to the result.

Subtract (1 << 4) from L, L becomes 2.
Subtract (1 << 4) from R, R becomes 5.

Now, Position of MSB in L is 1
Position of MSB in R is 2
Since positions are different all value (1 << i) for all
0   i   max(MSB1, MSB2)
i.e. Add ((1 << 2) + (1 << 1) + (1 << 0)) = 7
Hence, final result is 16 + 7 = 23.
```

Below is the implementation of above approach.

C++

```
 // C++ Program to find the bitwise
// OR of all the intgers in range L-R
#include <bits/stdc++.h>
using namespace std;

// Returns the Most Significant Bit
// Position (MSB)
int MSBPosition(long long int N)
{
    int msb_p = -1;
    while (N) {
        N = N >> 1;
        msb_p++;
    }
    return msb_p;
}

// Returns the Bitwise OR of all
// integers between L and R
long long int findBitwiseOR(long long int L,
                            long long int R)
{
    long long int res = 0;

    // Find the MSB position in L
    int msb_p1 = MSBPosition(L);
```

```
    // Find the MSB position in R
    int msb_p2 = MSBPosition(R);

    while (msb_p1 == msb_p2) {
        long long int res_val = (1 << msb_p1);

        // Add this value until msb_p1 and
        // msb_p2 are same;
        res += res_val;

        L -= res_val;
        R -= res_val;

        // Calculate msb_p1 and msb_p2
        msb_p1 = MSBPosition(L);
        msb_p2 = MSBPosition(R);
    }
    // Find the max of msb_p1 and msb_p2
    msb_p1 = max(msb_p1, msb_p2);

    // Set all the bits from msb_p1 upto
    // 0th bit in the result
    for (int i = msb_p1; i >= 0; i--) {
        long long int res_val = (1 << i);
        res += res_val;
    }
    return res;
}

// Driver Code
int main()
{
    int L = 12, R = 18;
    cout << findBitwiseOR(L, R) << endl;
    return 0;
}
```

Java

```
 // Java Program to find
// the bitwise OR of all
// the intgers in range L-R
import java.io.*;

class GFG
{
```

```
// Returns the Most Significant
// Bit Position (MSB)
static int MSBPosition(long N)
{
    int msb_p = -1;
    while (N > 0)
    {
        N = N >> 1;
        msb_p++;
    }
    return msb_p;
}

// Returns the Bitwise
// OR of all integers
// between L and R
static long findBitwiseOR(long L,
                          long R)
{
    long res = 0;

    // Find the MSB
    // position in L
    int msb_p1 = MSBPosition(L);

    // Find the MSB
    // position in R
    int msb_p2 = MSBPosition(R);

    while (msb_p1 == msb_p2)
    {
        long res_val = (1 << msb_p1);

        // Add this value until
        // msb_p1 and msb_p2 are same;
        res += res_val;

        L -= res_val;
        R -= res_val;

        // Calculate msb_p1
        // and msb_p2
        msb_p1 = MSBPosition(L);
        msb_p2 = MSBPosition(R);
    }

    // Find the max of
    // msb_p1 and msb_p2
```

```
    msb_p1 = Math.max(msb_p1,
                      msb_p2);

    // Set all the bits
    // from msb_p1 upto
    // 0th bit in the result
    for (int i = msb_p1; i >= 0; i--)
    {
        long res_val = (1 << i);
        res += res_val;
    }
    return res;
}

// Driver Code
public static void main (String[] args)
{
    int L = 12, R = 18;
    System.out.println(findBitwiseOR(L, R));
}
}

// This code is contributed
// by anuj_67.
```

C#

```
 // C# Program to find
// the bitwise OR of all
// the intgers in range L-R
using System;

class GFG
{

// Returns the Most Significant
// Bit Position (MSB)
static int MSBPosition(long N)
{
    int msb_p = -1;
    while (N > 0)
    {
        N = N >> 1;
        msb_p++;
    }
    return msb_p;
}
```

```
// Returns the Bitwise
// OR of all integers
// between L and R
static long findBitwiseOR(long L,
                          long R)
{
    long res = 0;

    // Find the MSB
    // position in L
    int msb_p1 = MSBPosition(L);

    // Find the MSB
    // position in R
    int msb_p2 = MSBPosition(R);

    while (msb_p1 == msb_p2)
    {
        long res_val = (1 << msb_p1);

        // Add this value until
        // msb_p1 and msb_p2 are same;
        res += res_val;

        L -= res_val;
        R -= res_val;

        // Calculate msb_p1
        // and msb_p2
        msb_p1 = MSBPosition(L);
        msb_p2 = MSBPosition(R);
    }

    // Find the max of
    // msb_p1 and msb_p2
    msb_p1 = Math.Max(msb_p1,
                      msb_p2);

    // Set all the bits
    // from msb_p1 upto
    // 0th bit in the result
    for (int i = msb_p1; i >= 0; i--)
    {
        long res_val = (1 << i);
        res += res_val;
    }
    return res;
}
```

```
// Driver Code
public static void Main ()
{
    int L = 12, R = 18;
    Console.WriteLine(findBitwiseOR(L, R));
}
}

// This code is contributed
// by anuj_67.
```

PHP

```
 <?php
// PHP Program to find the
// bitwise OR of all the
// intgers in range L-R

// Returns the Most Significant
// Bit Position (MSB)
function MSBPosition($N)
{
    $msb_p = -1;
    while ($N)
    {
        $N = $N >> 1;
        $msb_p++;
    }
    return $msb_p;
}

// Returns the Bitwise
// OR of all integers
// between L and R
function findBitwiseOR($L, $R)
{
    $res = 0;

    // Find the MSB
    // position in L
    $msb_p1 = MSBPosition($L);

    // Find the MSB
    // position in R
    $msb_p2 = MSBPosition($R);

    while ($msb_p1 == $msb_p2)
```

```
    {
        $res_val = (1 << $msb_p1);

        // Add this value until
        // msb_p1 and msb_p2 are same;
        $res += $res_val;

        $L -= $res_val;
        $R -= $res_val;

        // Calculate msb_p1
        // and msb_p2
        $msb_p1 = MSBPosition($L);
        $msb_p2 = MSBPosition($R);
    }

    // Find the max of
    // msb_p1 and msb_p2
    $msb_p1 = max($msb_p1,
                  $msb_p2);

    // Set all the bits from msb_p1
    // upto 0th bit in the result
    for ($i = $msb_p1; $i >= 0; $i--)
    {
        $res_val = (1 << $i);
        $res += $res_val;
    }
    return $res;
}

// Driver Code
$L = 12; $R = 18;
echo findBitwiseOR($L, $R);

// This code is contributed
// by anuj_67.
?>
```

Output:

```
31
```

Time Complexity: O(N), where N is the Most significant bit.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/bitwise-or-or-of-a-range/

Chapter 20

Bitwise Operators in C/C++

Bitwise Operators in C/C++ - GeeksforGeeks

In C, following 6 operators are bitwise operators (work at bit-level)

& (bitwise AND) Takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.

| (bitwise OR) Takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 any of the two bits is 1.

ˆ (bitwise XOR) Takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.

<< (left shift) Takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.

>> (right shift) Takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.

~ (bitwise NOT) Takes one number and inverts all bits of it

Following is example C program.

```
 /* C Program to demonstrate use of bitwise operators */
#include<stdio.h>
int main()
{
    unsigned char a = 5, b = 9; // a = 5(00000101), b = 9(00001001)
    printf("a = %d, b = %d\n", a, b);
    printf("a&b = %d\n", a&b); // The result is 00000001
    printf("a|b = %d\n", a|b);  // The result is 00001101
    printf("a^b = %d\n", a^b); // The result is 00001100
    printf("~a = %d\n", a = ~a);   // The result is 11111010
    printf("b<<1 = %d\n", b<<1);  // The result is 00010010
    printf("b>>1 = %d\n", b>>1);  // The result is 00000100
```

```
    return 0;
}
```

Output:

```
a = 5, b = 9
a&b = 1
a|b = 13
a^b = 12
~a = 250
b<<1 = 18
b>>1 = 4
```

Following are interesting facts about bitwise operators.

1) The left shift and right shift operators should not be used for negative numbers If any of the operands is a negative number, it results in undefined behaviour. For example results of both -1 << 1 and 1 << -1 is undefined. Also, if the number is shifted more than the size of integer, the behaviour is undefined. For example, 1 << 33 is undefined if integers are stored using 32 bits. See this for more details.

2) The bitwise XOR operator is the most useful operator from technical interview perspective. It is used in many problems. A simple example could be "Given a set of numbers where all elements occur even number of times except one number, find the odd occurring number" This problem can be efficiently solved by just doing XOR of all numbers.

```
 // Function to return the only odd occurring element
int findOdd(int arr[], int n) {
   int res = 0, i;
   for (i = 0; i < n; i++)
     res ^= arr[i];
   return res;
}

int main(void) {
   int arr[] = {12, 12, 14, 90, 14, 14, 14};
   int n = sizeof(arr)/sizeof(arr[0]);
   printf ("The odd occurring element is %d ", findOdd(arr, n));
   return 0;
}
// Output: The odd occurring element is 90
```

The following are many other interesting problems which can be used using XOR operator. Find the Missing Number, swap two numbers without using a temporary variable, A Memory Efficient Doubly Linked List, and Find the two non-repeating elements. There are many more (See this, this, this, this, this and this)

3) The bitwise operators should not be used in place of logical operators.
The result of logical operators (&&, || and !) is either 0 or 1, but bitwise operators return an

integer value. Also, the logical operators consider any non-zero operand as 1. For example, consider the following program, the results of & and && are different for same operands.

```
 int main()
{
   int x = 2, y = 5;
   (x & y)? printf("True ") : printf("False ");
   (x && y)? printf("True ") : printf("False ");
   return 0;
}
// Output: False True
```

4) The left-shift and right-shift operators are equivalent to multiplication and division by 2 respectively.
As mentioned in point 1, it works only if numbers are positive.

```
 int main()
{
   int x = 19;
   printf ("x << 1 = %d\n", x << 1);
   printf ("x >> 1 = %d\n", x >> 1);
   return 0;
}
// Output: 38 9
```

5) The & operator can be used to quickly check if a number is odd or even
The value of expression (x & 1) would be non-zero only if x is odd, otherwise the value would be zero.

```
 int main()
{
   int x = 19;
   (x & 1)? printf("Odd"): printf("Even");
   return 0;
}
// Output: Odd
```

6) The ~ operator should be used carefully
The result of ~ operator on a small number can be a big number if the result is stored in an unsigned variable. And result may be negative number if result is stored in signed variable (assuming that the negative numbers are stored in 2's complement form where leftmost bit is the sign bit)

```
 // Note that the output of the following program is compiler dependent
int main()
{
```

```
   unsigned int x = 1;
   printf("Signed Result %d \n", ~x);
   printf("Unsigned Result %ud \n", ~x);
   return 0;
}
/* Output:
Signed Result -2
Unsigned Result 4294967294d */
```

Important Links :

1. Bits manipulation (Important tactics)
2. Bitwise Hacks for Competitive Programming
3. Bit Tricks for Competitive Programming

Improved By : Shubham Dhiman 1, prakash_

Source

https://www.geeksforgeeks.org/bitwise-operators-in-c-cpp/

Chapter 21

Bitwise Sieve

Bitwise Sieve - GeeksforGeeks

Given a number n, print all primes smaller than n.

Examples :

```
Input : 30
Output : 2 3 5 7 11 13 17 19 23 29

Input : n = 100
Output : 2 3 5 7 11 13 17 19 23 29 31 37
         41 43 47 53 59 61 67 71 73 79 83
         89 97
```

We know how to calculate all primes less than n by Sieve of Eratosthenes. Below is an implementation of Sieve.
One optimization in below implementation is, we have skipped all even numbers altogether.
We reduce size of prime array to half. We also reduce all iterations to half.

C++

```
 // C++ program to implement normal Sieve
// of Eratosthenes using simple optimization
// to reduce size of prime array to half and
// reducing iterations.
#include <bits/stdc++.h>
using namespace std;

void normalSieve(int n)
{
```

```
    // prime[i] is going to store true if
    // if i*2 + 1 is composite.
    bool prime[n/2];
    memset(prime, false, sizeof(prime));

    // 2 is the only even prime so we can
    // ignore that. Loop starts from 3.
    for (int i=3 ; i*i < n; i+=2)
    {
        // If i is prime, mark all its
        // multiples as composite
        if (prime[i/2] == false)
            for (int j=i*i; j<n; j+=i*2)
                prime[j/2] = true;
    }

    // writing 2 separately
    printf("2 ");

    // Printing other primes
    for (int i=3; i<n ; i+=2)
        if (prime[i/2] == false)
            printf( "%d " , i );
}

// Driver code
int main()
{
    int n = 100 ;
    normalSieve(n);
    return 0;
}
```

Java

```
 // Java program to implement normal Sieve
// of Eratosthenes using simple optimization
// to reduce size of prime array to half and
// reducing iterations.
import java.util.Arrays;

class GFG
{
    static void normalSieve(int n)
    {
        // prime[i] is going to store true if
        // if i*2 + 1 is composite.
        boolean prime[]=new boolean[n / 2];
```

```
        Arrays.fill(prime, false);

        // 2 is the only even prime so we can
        // ignore that. Loop starts from 3.
        for (int i = 3 ; i * i < n; i += 2)
        {
            // If i is prime, mark all its
            // multiples as composite
            if (prime[i / 2] == false)
                for (int j = i * i; j < n; j += i * 2)
                    prime[j / 2] = true;
        }

        // writing 2 separately
        System.out.print("2 ");

        // Printing other primes
        for (int i = 3; i < n ; i += 2)
            if (prime[i / 2] == false)
                System.out.print(i + " ");
    }
    public static void main (String[] args)
    {
        int n = 100 ;
        normalSieve(n);
    }
}

// This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to implement normal Sieve
// of Eratosthenes using simple optimization
// to reduce size of prime array to half and
// reducing iterations.
using System;

namespace prime
{
    public class GFG
    {

        public static void normalSieve(int n)
        {

        // prime[i] is going to store true if
        // if i*2 + 1 is composite.
```

```
        bool[] prime = new bool[n/2];

        for(int i = 0; i < n/2; i++)
            prime[i] = false;

        // 2 is the only even prime so we can
        // ignore that. Loop starts from 3.
        for(int i = 3; i*i < n; i = i+2)
        {
            // If i is prime, mark all its
            // multiples as composite
            if (prime[i / 2] == false)

                for (int j = i * i; j < n; j += i * 2)
                    prime[j / 2] = true;
        }

        // writing 2 separately
        Console.Write("2 ");

        // Printing other primes
        for (int i = 3; i < n ; i += 2)

            if (prime[i / 2] == false)
                Console.Write(i + " ");

        }

        // Driver Code
        public static void Main()
        {
        int n = 100;
        normalSieve(n);
        }
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to implement normal
// Sieve of Eratosthenes using
// simple optimization to reduce
// size of prime array to half and
// reducing iterations.
function normalSieve($n)
```

```
{
    // prime[i] is going to store
    // true if if i*2 + 1 is composite.
    $prime = array_fill(0, (int)($n / 2),
                                  false);

    // 2 is the only even prime so
    // we can ignore that. Loop
    // starts from 3.
    for ($i = 3 ; $i * $i < $n; $i += 2)
    {
        // If i is prime, mark all its
        // multiples as composite
        if ($prime[$i / 2] == false)
            for ($j = $i * $i; $j < $n;
                          $j += $i * 2)
                $prime[$j / 2] = true;
    }

    // writing 2 separately
    echo "2 ";

    // Printing other primes
    for ($i = 3; $i < $n ; $i += 2)
        if ($prime[$i / 2] == false)
            echo $i . " ";
}

// Driver code
$n = 100 ;
normalSieve($n);

// This code is contributed by mits.
?>
```

Output :

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Further optimization using bitwise operators.
The above implementation uses bool data type which takes 1 byte. We can optimize space to n/8 by using individual bits of an integer to represent individual primes. We create an integer array of size n/64. Note that the size of array is reduced to n/64 from n/2 (Assuming that integers take 32 bits).

C++

```
 // C++ program to implement bitwise Sieve
// of Eratosthenes.
#include <bits/stdc++.h>
using namespace std;

// Checks whether x is prime or composite
bool ifnotPrime(int prime[], int x)
{
    // checking whether the value of element
    // is set or not. Using prime[x/64], we find
    // the slot in prime array. To find the bit
    // number, we divide x by 2 and take its mod
    // with 32.
    return (prime[x/64] & (1 << ((x >> 1) & 31)));
}

// Marks x composite in prime[]
bool makeComposite(int prime[], int x)
{
    // Set a bit corresponding to given element.
    // Using prime[x/64], we find the slot in prime
    // array. To find the bit number, we divide x
    // by 2 and take its mod with 32.
    prime[x/64] |= (1 << ((x >> 1) & 31));
}

// Prints all prime numbers smaller than n.
void bitWiseSieve(int n)
{
    // Assuming that n takes 32 bits, we reduce
    // size to n/64 from n/2.
    int prime[n/64];

    // Initializing values to 0 .
    memset(prime, 0, sizeof(prime));

    // 2 is the only even prime so we can ignore that
    // loop starts from 3 as we have used in sieve of
    // Eratosthenes .
    for (int i = 3; i * i <= n; i += 2) {

        // If i is prime, mark all its multiples as
        // composite
        if (!ifnotPrime(prime, i))
            for (int j = i * i, k = i << 1; j < n; j += k)
                makeComposite(prime, j);
    }
```

```
    // writing 2 separately
    printf("2 ");

    // Printing other primes
    for (int i = 3; i <= n; i += 2)
        if (!ifnotPrime(prime, i))
            printf("%d ", i);
}

// Driver code
int main()
{
    int n = 30;
    bitWiseSieve(n);
    return 0;
}
```

Java

```
 // JAVA Code to implement Bitwise
// Sieve of Eratosthenes.
import java.util.*;

class GFG {

    // Checks whether x is prime or composite
    static int ifnotPrime(int prime[], int x)
    {
        // checking whether the value of element
        // is set or not. Using prime[x/64],
        // we find the slot in prime array.
        // To find the bit number, we divide x
        // by 2 and take its mod with 32.
        return (prime[x/64] & (1 << ((x >> 1) & 31)));
    }

    // Marks x composite in prime[]
    static void makeComposite(int prime[], int x)
    {
        // Set a bit corresponding to given element.
        // Using prime[x/64], we find the slot
        // in prime array. To find the bit number,
        // we divide x by 2 and take its mod with 32.
        prime[x/64] |= (1 << ((x >> 1) & 31));
    }

    // Prints all prime numbers smaller than n.
    static void bitWiseSieve(int n)
```

```
    {
        // Assuming that n takes 32 bits,
        // we reduce size to n/64 from n/2.
        int prime[] = new int[n/64 + 1];


        // 2 is the only even prime so we
        // can ignore that loop starts from
        // 3 as we have used in sieve of
        // Eratosthenes .
        for (int i = 3; i * i <= n; i += 2) {

            // If i is prime, mark all its
            // multiples as composite
            if (ifnotPrime(prime, i)==0)
                for (int j = i * i, k = i << 1;
                                    j < n; j += k)
                    makeComposite(prime, j);
        }

        // writing 2 separately
        System.out.printf("2 ");

        // Printing other primes
        for (int i = 3; i <= n; i += 2)
            if (ifnotPrime(prime, i) == 0)
                System.out.printf("%d ", i);
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int n = 30;
        bitWiseSieve(n);
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

C#

```
// C# Code to implement Bitwise
// Sieve of Eratosthenes.
using System;

class GFG
{

// Checks whether x is
// prime or composite
```

```
static int ifnotPrime(int[] prime, int x)
{
// checking whether the value
// of element is set or not.
// Using prime[x/64], we find
// the slot in prime array.
// To find the bit number, we
// divide x by 2 and take its
// mod with 32.
return (prime[x / 64] &
(1 << ((x >> 1) & 31)));
}

// Marks x composite in prime[]
static void makeComposite(int[] prime,
int x)
{
// Set a bit corresponding to
// given element. Using prime[x/64],
// we find the slot in prime array.
// To find the bit number, we divide
// x by 2 and take its mod with 32.
prime[x / 64] |= (1 << ((x >> 1) & 31));
}

// Prints all prime numbers
// smaller than n.
static void bitWiseSieve(int n)
{
// Assuming that n takes 32 bits,
// we reduce size to n/64 from n/2.
int[] prime = new int[(int)(n / 64) + 1];

// 2 is the only even prime so we
// can ignore that loop starts from
// 3 as we have used in sieve of
// Eratosthenes .
for (int i = 3; i * i <= n; i += 2) { // If i is prime, mark all its // multiples as composite if (ifnotPrime(prime, i) == 0) for (int j = i * i, k = i << 1; j < n; j += k) makeComposite(prime, j); } // writing 2 separately Console.Write("2 "); // Printing other primes for (int i = 3; i <= n; i += 2) if (ifnotPrime(prime, i) == 0) Console.Write(i + " "); } // Driver Code static void Main() { int n = 30; bitWiseSieve(n); } } // This code is contributed by mits [tabby title="PHP"]


 <?php
// PHP program to implement
// bitwise Sieve of Eratosthenes.
$prime;
```

```
// Checks whether x is
// prime or composite
function ifnotPrime($x)
{
    global $prime;

    // checking whether the value
    // of element is set or not.
    // Using prime[x/64], we find
    // the slot in prime array.
    // To find the bit number, we
    // divide x by 2 and take its
    // mod with 32.
    return $a = ($prime[(int)($x / 64)] &
                        (1 << (($x >> 1) & 31)));
}

// Marks x composite in prime[]
function makeComposite($x)
{
    global $prime;

    // Set a bit corresponding to
    // given element. Using prime[x/64],
    // we find the slot in prime
    // array. To find the bit number,
    // we divide x by 2 and take its
    // mod with 32.
    $prime[(int)($x / 64)] |=
                (1 << (($x >> 1) & 31));
}

// Prints all prime
// numbers smaller than n.
function bitWiseSieve($n)
{
    global $prime;

    // Assuming that n takes
    // 32 bits, we reduce
    // size to n/64 from n/2.
    // Initializing values to 0 .
    $prime = array_fill(0,
                   (int)ceil($n / 64), 0);

    // 2 is the only even prime
    // so we can ignore that
    // loop starts from 3 as we
```

```
    // have used in sieve of
    // Eratosthenes .
    for ($i = 3; $i * $i <= $n; $i += 2)
    {

        // If i is prime, mark
        // all its multiples as
        // composite
        if (!ifnotPrime($i))
            for ($j = $i * $i,
                 $k = $i << 1;
                 $j < $n; $j += $k)
                makeComposite($j);
    }

    // writing 2 separately
    echo "2 ";

    // Printing other primes
    for ($i = 3; $i <= $n; $i += 2)
        if (!ifnotPrime($i))
            echo $i." ";
}

// Driver code
$n = 30;
bitWiseSieve($n);

// This code is contributed
// by mits.
?>
```

Output:

```
2 3 5 7 11 13 17 19 23 29
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/bitwise-sieve/

Chapter 22

Bitwise and (or &) of a range

Bitwise and (or &) of a range - GeeksforGeeks

Given two non-negative long integers, x and y given x <= y, the task is to find bit-wise and of all integers from x and y, i.e., we need to compute value of x & (x+1) & … & (y-1) & y.7

Examples:

```
Input  : x = 12, y = 15
Output : 12
12 & 13 & 14 & 14 = 12

Input  : x = 10, y = 20
Output : 0
```

A **simple solution** is to traverse all numbers from x to y and do bit-wise and of all numbers in range.

An **efficient solution** is to follow following steps.
1) Find position of Most Significant Bit (MSB) in both numbers.
2) If positions of MSB are different, then result is 0.
3) If positions are same. Let positions be msb_p.
……a) We add 2^{msb_p} to result.
……b) We subtract 2^{msb_p} from x and y,
……c) Repeat steps 1, 2 and 3 for new values of x and y.

```
Example 1 :
x = 10, y = 20
Result is initially 0.
Position of MSB in x = 3
Position of MSB in y = 4
Since positions are different, return result.
```

```
Example 2 :
x = 17, y = 19
Result is initially 0.
Position of MSB in x = 4
Position of MSB in y = 4
Since positions are same, we compute 24.

We add 24 to result.
Result becomes 16.

We subtract this value from x and y.
New value of x  = x - 24  = 17 - 16 = 1
New value of y  = y - 24  = 19 - 16 = 3

Position of MSB in new x = 1
Position of MSB in new y = 2
Since positions are different, we return result.
```

C++

```
 // An efficient C++ program to find bit-wise & of all
// numbers from x to y.
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;

// Find position of MSB in n. For example if n = 17,
// then position of MSB is 4. If n = 7, value of MSB
// is 3
int msbPos(ll n)
{
    int msb_p = -1;
    while (n)
    {
        n = n>>1;
        msb_p++;
    }
    return msb_p;
}

// Function to find Bit-wise & of all numbers from x
// to y.
ll andOperator(ll x, ll y)
{
    ll res = 0; // Initialize result
```

```
    while (x && y)
    {
        // Find positions of MSB in x and y
        int msb_p1 = msbPos(x);
        int msb_p2 = msbPos(y);

        // If positions are not same, return
        if (msb_p1 != msb_p2)
            break;

        // Add 2^msb_p1 to result
        ll msb_val =  (1 << msb_p1);
        res = res + msb_val;

        // subtract 2^msb_p1 from x and y.
        x = x - msb_val;
        y = y - msb_val;
    }

    return res;
}

// Driver code
int main()
{
    ll x = 10, y = 15;
    cout << andOperator(x, y);
    return 0;
}
```

Java

```
 // An efficient Java program to find bit-wise
// & of all numbers from x to y.
class GFG {

    // Find position of MSB in n. For example
    // if n = 17, then position of MSB is 4.
    // If n = 7, value of MSB is 3
    static int msbPos(long n)
    {

        int msb_p = -1;
        while (n > 0) {
            n = n >> 1;
            msb_p++;
        }
```

```
        return msb_p;
    }

    // Function to find Bit-wise & of all
    // numbers from x to y.
    static long andOperator(long x, long y)
    {

        long res = 0; // Initialize result

        while (x > 0 && y > 0) {

            // Find positions of MSB in x and y
            int msb_p1 = msbPos(x);
            int msb_p2 = msbPos(y);

            // If positions are not same, return
            if (msb_p1 != msb_p2)
                break;

            // Add 2^msb_p1 to result
            long msb_val = (1 << msb_p1);
            res = res + msb_val;

            // subtract 2^msb_p1 from x and y.
            x = x - msb_val;
            y = y - msb_val;
        }

        return res;
    }

    // Driver code
    public static void main(String[] args)
    {

        long x = 10, y = 15;

        System.out.print(andOperator(x, y));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # An efficient Python program to find
```

```
# bit-wise & of all numbers from x to y.

# Find position of MSB in n. For example
# if n = 17, then position of MSB is 4.
# If n = 7, value of MSB is 3
def msbPos(n):

    msb_p = -1
    while (n > 0):

        n = n >> 1
        msb_p += 1

    return msb_p

# Function to find Bit-wise & of
# all numbers from x to y.
def andOperator(x, y):

    res = 0 # Initialize result

    while (x > 0 and y > 0):

        # Find positions of MSB in x and y
        msb_p1 = msbPos(x)
        msb_p2 = msbPos(y)

        # If positions are not same, return
        if (msb_p1 != msb_p2):
            break

        # Add 2^msb_p1 to result
        msb_val = (1 << msb_p1)
        res = res + msb_val

        # subtract 2^msb_p1 from x and y.
        x = x - msb_val
        y = y - msb_val

    return res

# Driver code
x, y = 10, 15
print(andOperator(x, y))

# This code is contributed by Anant Agarwal.
```

C#

```
 // An efficient C# program to find bit-wise & of all
// numbers from x to y.
using System;

class GFG
{
    // Find position of MSB in n.
    // For example if n = 17,
    // then position of MSB is 4.
    // If n = 7, value of MSB
    // is 3
    static int msbPos(long n)
    {
        int msb_p = -1;
        while (n > 0)
        {
            n = n >> 1;
            msb_p++;
        }
        return msb_p;
    }

    // Function to find Bit-wise
    // & of all numbers from x
    // to y.
    static long andOperator(long x, long y)
    {
        // Initialize result
        long res = 0;

        while (x > 0 && y > 0)
        {
            // Find positions of MSB in x and y
            int msb_p1 = msbPos(x);
            int msb_p2 = msbPos(y);

            // If positions are not same, return
            if (msb_p1 != msb_p2)
                break;

            // Add 2^msb_p1 to result
            long msb_val = (1 << msb_p1);
            res = res + msb_val;

            // subtract 2^msb_p1 from x and y.
            x = x - msb_val;
            y = y - msb_val;
        }
```

```
        return res;
    }

    // Driver code
    public static void Main()
    {
        long x = 10, y = 15;
        Console.WriteLine(andOperator(x, y));
    }
}

// This code is contributed by Anant Agarwal.
```

Output:

```
8
```

More efficient solution

1. Flip the LSB of b.
2. And check if the new number is in range(a < number < b) or not

- if the number greater than 'a' again flip lsb
- if it is not then that's the answer

NOTE: We are not considering inclusive range so if the number is 'a' we shouldn't flip anymore i.e.'a' will be the answer.

```
Example 1 :
x = 10, y = 20
y = 10100
Flip LSB of y New Number = 10000 i.e. 16
16 is in range so Again flip : 00000 i.e. 0 and 0 is not in range so 0 is the answer.

Example 2 :
x = 17, y = 19
y = 10011
Flip LSB of y so New Number = 10010 i.e. 18
18 is in range so Again flip : 10000 i.e. 16 and 16 is 'a' so Stop , answer is 16.
```

```
 // An efficient C++ program to find bit-wise & of all
// numbers from x to y.
#include <bits/stdc++.h>
#define ll long long
```

```
using namespace std;
ll andOperator(ll a, ll b){
while(a < b){
    // -b is the 2's complement of b when do bitwise or with b
    //we get LSB and we subtract that from b
    b -= (b & -b);
}
return b;
}
// Driver code
int main(){
    ll a, b;
    a = 10; b = 15;
    cout << andOperator(a, b);
}
```

Output:

```
8
```

Improved By : Bibhu Pala

Source

https://www.geeksforgeeks.org/bitwise-and-or-of-a-range/

Chapter 23

Bitwise recursive addition of two integers

Bitwise recursive addition of two integers - GeeksforGeeks

When adding two binary numbers by hand we keep the carry bits in mind and add it at the same time. But to do same thing in program we need a lot of checks. Recursive solution can be imagined as addition of *carry* and $a \hat{} b$ (two inputs) until *carry* becomes 0.

Examples :

```
Input : int x = 45, y = 45
Output : 90

Input : int x = 4, y = 78
Output : 82
```

Sum of two bits can be obtained by performing XOR (^) of the two bits. Carry bit can be obtained by performing AND (&) of two bits.
Above is simple Half Adder logic that can be used to add 2 single bits. We can extend this logic for integers. If x and y don't have set bits at same position(s), then bitwise XOR (^) of x and y gives the sum of x and y. To incorporate common set bits also, bitwise AND (&) is used. Bitwise AND of x and y gives all carry bits. We calculate (x & y) << 1 and add it to x ^ y to get the required result.

One important observation is, if (x & y) becomes 0, then result is x ^ y.

C

```
 // CPP program to do recursive addition
// of two integers
#include <stdio.h>
```

```
int add(int x, int y) {
    int keep = (x & y) << 1;
    int res = x^y;

    // If bitwise & is 0, then there
    // is not going to be any carry.
    // Hence result of XOR is addition.
    if (keep == 0)
        return res;

    add(keep, res);
}

// Driver code
int main(){
    printf("%d", add(15, 38));
    return 0;
}
```

Java

```
 // Java program to do recursive addition
// of two integers
import java.io.*;

class GFG {

    static int add(int x, int y)
    {
        int keep = (x & y) << 1;
        int res = x^y;

        // If bitwise & is 0, then there
        // is not going to be any carry.
        // Hence result of XOR is addition.
        if (keep == 0)
            return res;

        return add(keep, res);
    }

    // Driver code
    public static void main (String[] args)
    {
        System.out.println(add(15, 38));
    }
}
```

```
// This code is contributed by Ajit.
```

C#

```
 // C# program to do recursive
// addition of two integers
using System;

class GFG {

    static int add(int x, int y)
    {
        int keep = (x & y) << 1;
        int res = x^y;

        // If bitwise & is 0, then there
        // is not going to be any carry.
        // Hence result of XOR is addition.
        if (keep == 0)
            return res;

        return add(keep, res);
    }

    // Driver code
    public static void Main ()
    {
        Console.Write(add(15, 38));
    }
}

// This code is contributed by Smitha.
```

PHP

```
 <?php
// php program to do recursive addition
// of two integers

function add($x, $y) {
    $keep = ($x & $y) << 1;
    $res = $x^$y;

    // If bitwise & is 0, then there
    // is not going to be any carry.
    // Hence result of XOR is addition.
    if ($keep == 0)
```

```
    {
        echo $res;
        exit(0);
    }

    add($keep, $res);
}

// Driver code
$k= add(15, 38);

// This code is contributed by mits.
?>
```

Output:

```
53
```

Improved By : jit_t, Smitha Dinesh Semwal, Mithun Kumar

Source

https://www.geeksforgeeks.org/bitwise-recursive-addition-two-integers/

Chapter 24

Bitwise right shift operators in Java

Bitwise right shift operators in Java - GeeksforGeeks

In C/C++ there is only one right shift operator '>>' which should be used only for positive integers or unsigned integers. Use of right shift operator for negative numbers is not recommended in C/C++, and when used for negative numbers, output is compiler dependent (See this). Unlike C++, Java supports following two right shift operators.

1) >> (Signed right shift) In Java, the operator '>>' is signed right shift operator. All integers are signed in Java, and it is fine to use >> for negative numbers. The operator '>>' uses the sign bit (left most bit) to fill the trailing positions after shift. If the number is negative, then 1 is used as a filler and if the number is positive, then 0 is used as a filler. For example, if binary representation of number is **1**0....100, then right shifting it by 2 using >> will make it **11**.....1.
See following Java programs as example '>>'

```
 class Test {
    public static void main(String args[])  {
       int x = -4;
       System.out.println(x>>1);
       int y = 4;
       System.out.println(y>>1);
    }
}
```

Output:

```
-2
2
```

2) >>> (Unsigned right shift) In Java, the operator '>>>' is unsigned right shift operator. It always fills 0 irrespective of the sign of the number.

```
class Test {
   public static void main(String args[])  {

      // x is stored using 32 bit 2's complement form.
      // Binary representation of -1 is all 1s (111..1)
      int x = -1;

      System.out.println(x>>>29);  // The value of 'x>>>29' is 00...0111
      System.out.println(x>>>30);  // The value of 'x>>>30' is 00...0011
      System.out.println(x>>>31);  // The value of 'x>>>31' is 00...0001
   }
}
```

Output:

```
7
3
1
```

Improved By : nagen010

Source

https://www.geeksforgeeks.org/bitwise-shift-operators-in-java/

Chapter 25

Booth's Multiplication Algorithm

Booth's Multiplication Algorithm - GeeksforGeeks

Booth's algorithm is a multiplication algorithm that multiplies two signed binary numbers in 2's compliment notation.
Booth used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed. Booth's algorithm is of interest in the study of computer architecture. Here's the implememtation of the algorithm.

Examples:

```
Input : 0110, 0010
Output :  qn      q[n+1]                  AC      QR     sc(step count)
                          initial         0000   0010        4
          0       0       rightShift      0000   0001        3
          1       0       A = A - BR      1010
                          rightShift      1101   0000        2
          0       1       A = A + BR      0011
                          rightShift      0001   1000        1
          0       0       rightShift      0000   1100        0

Result=1100
```

Algorithm :

Put multiplicand in BR and multiplier in QR
and then the algorithm works as per the following conditions :
1. If Q_n and Q_{n+1} are same i.e. 00 or 11 perform arithematic shift by 1 bit.
2. If Q_n Q_{n+1} = 10 do A= A + BR and perform arithematic shift by 1 bit.
3. If Q_n Q_{n+1} = 01 do A= A – BR and perform arithematic shift by 1 bit.

C++

```
 // CPP code to implement booth's algorithm
#include <bits/stdc++.h>

using namespace std;

// function to perform adding in the accumulator
void add(int ac[], int x[], int qrn)
{
    int i, c = 0;

    for (i = 0; i < qrn; i++) {

        // updating accumulator with A = A + BR
        ac[i] = ac[i] + x[i] + c;

        if (ac[i] > 1) {
            ac[i] = ac[i] % 2;
            c = 1;
        }
        else
            c = 0;
    }
}

// function to find the number's complement
void complement(int a[], int n)
{
    int i;
    int x[8] = { 0};
    x[0] = 1;

    for (i = 0; i < n; i++) {
        a[i] = (a[i] + 1) % 2;
    }
    add(a, x, n);
}

// function ro perform right shift
void rightShift(int ac[], int qr[], int& qn, int qrn)
{
    int temp, i;
    temp = ac[0];
    qn = qr[0];

    cout << "\t\trightShift\t";
```

```
    for (i = 0; i < qrn - 1; i++) {
        ac[i] = ac[i + 1];
        qr[i] = qr[i + 1];
    }
    qr[qrn - 1] = temp;
}

// function to display oprations
void display(int ac[], int qr[], int qrn)
{
    int i;

    // accumulator content
    for (i = qrn - 1; i >= 0; i--)
        cout << ac[i];
    cout << "\t";

    // multiplier content
    for (i = qrn - 1; i >= 0; i--)
        cout << qr[i];
}

// Function to implement booth's algo
void boothAlgorithm(int br[], int qr[], int mt[], int qrn, int sc)
{

    int qn = 0, ac[10] = { 0 };
    int temp = 0;
    cout << "qn\tq[n+1]\t\tBR\t\tAC\tQR\t\tsc\n";
    cout << "\t\t\tinitial\t\t";

    display(ac, qr, qrn);
    cout << "\t\t" << sc << "\n";

    while (sc != 0) {
        cout << qr[0] << "\t" << qn;

        // SECOND CONDITION
        if ((qn + qr[0]) == 1)
        {
            if (temp == 0) {

                // subtract BR from accumulator
                add(ac, mt, qrn);
                cout << "\t\tA = A - BR\t";

                for (int i = qrn - 1; i >= 0; i--)
                    cout << ac[i];
```

```
                temp = 1;
            }

            // THIRD CONDITION
            else if (temp == 1)
            {
                // add BR to accumulator
                add(ac, br, qrn);
                cout << "\t\tA = A + BR\t";

                for (int i = qrn - 1; i >= 0; i--)
                    cout << ac[i];
                temp = 0;
            }
            cout << "\n\t";
            rightShift(ac, qr, qn, qrn);
        }

        // FIRST CONDITION
        else if (qn - qr[0] == 0)
            rightShift(ac, qr, qn, qrn);

        display(ac, qr, qrn);

        cout << "\t";

        // decrement counter
        sc--;
        cout << "\t" << sc << "\n";
    }
}

// driver code
int main(int argc, char** arg)
{

    int mt[10], sc;
    int brn, qrn;

    // Number of multiplicand bit
    brn = 4;

    // multiplicand
    int br[] = { 0, 1, 1, 0 };

    // copy multiplier to temp array mt[]
    for (int i = brn - 1; i >= 0; i--)
        mt[i] = br[i];
```

```
    reverse(br, br + brn);

    complement(mt, brn);

    // No. of multiplier bit
    qrn = 4;

    // sequence counter
    sc = qrn;

    // multiplier
    int qr[] = { 1, 0, 1, 0 };
    reverse(qr, qr + qrn);

    boothAlgorithm(br, qr, mt, qrn, sc);

    cout << endl
         << "Result = ";

    for (int i = qrn - 1; i >= 0; i--)
        cout << qr[i];
}
```

C#

```
 // C# code to implement
// booth's algorithm
using System;

class GFG
{
    // function to perform
    // adding in the accumulator
    static void add(int []ac,
                    int []x,
                    int qrn)
    {
        int i, c = 0;

        for (i = 0; i < qrn; i++)
        {

            // updating accumulator
            // with A = A + BR
            ac[i] = ac[i] + x[i] + c;

            if (ac[i] > 1)
```

```
        {
            ac[i] = ac[i] % 2;
            c = 1;
        }
        else
            c = 0;
    }
}

// function to find
// the number's complement
static void complement(int []a, int n)
{
    int i;
    int []x = new int[8];
    Array.Clear(x, 0, 8);

    x[0] = 1;

    for (i = 0; i < n; i++)
    {
        a[i] = (a[i] + 1) % 2;
    }
    add(a, x, n);
}

// function to perform
// right shift
static void rightShift(int []ac, int []qr,
                       ref int qn, int qrn)
{
    int temp, i;
    temp = ac[0];
    qn = qr[0];

    Console.Write("\t\trightShift\t");

    for (i = 0; i < qrn - 1; i++)
    {
        ac[i] = ac[i + 1];
        qr[i] = qr[i + 1];
    }
    qr[qrn - 1] = temp;
}

// function to display
// operations
static void display(int []ac,
```

```
                    int []qr,
                    int qrn)
{
    int i;

    // accumulator content
    for (i = qrn - 1; i >= 0; i--)
        Console.Write(ac[i]);
    Console.Write("\t");

    // multiplier content
    for (i = qrn - 1; i >= 0; i--)
        Console.Write(qr[i]);
}

// Function to implement
// booth's algo
static void boothAlgorithm(int []br, int []qr,
                           int []mt, int qrn,
                           int sc)
{

    int qn = 0;
    int []ac = new int[10];
    Array.Clear(ac, 0, 10);

    int temp = 0;
    Console.Write("qn\tq[n + 1]\tBR\t" +
                  "\tAC\tQR\t\tsc\n");
    Console.Write("\t\t\tinitial\t\t");

    display(ac, qr, qrn);
    Console.Write("\t\t" + sc + "\n");

    while (sc != 0)
    {
        Console.Write(qr[0] + "\t" + qn);

        // SECOND CONDITION
        if ((qn + qr[0]) == 1)
        {
            if (temp == 0)
            {

                // subtract BR
                // from accumulator
                add(ac, mt, qrn);
                Console.Write("\t\tA = A - BR\t");
```

```
                for (int i = qrn - 1; i >= 0; i--)
                    Console.Write(ac[i]);
                temp = 1;
            }

            // THIRD CONDITION
            else if (temp == 1)
            {
                // add BR to accumulator
                add(ac, br, qrn);
                Console.Write("\t\tA = A + BR\t");

                for (int i = qrn - 1; i >= 0; i--)
                    Console.Write(ac[i]);
                temp = 0;
            }
            Console.Write("\n\t");
            rightShift(ac, qr, ref qn, qrn);
        }

        // FIRST CONDITION
        else if (qn - qr[0] == 0)
            rightShift(ac, qr,
                       ref qn, qrn);

        display(ac, qr, qrn);

        Console.Write("\t");

        // decrement counter
        sc--;
        Console.Write("\t" + sc + "\n");
    }
}

// Driver code
static void Main()
{

    int []mt = new int[10];
    int sc, brn, qrn;

    // Number of
    // multiplicand bit
    brn = 4;

    // multiplicand
```

```
        int []br = new int[]{ 0, 1, 1, 0 };

        // copy multiplier
        // to temp array mt[]
        for (int i = brn - 1; i >= 0; i--)
            mt[i] = br[i];

        Array.Reverse(br);

        complement(mt, brn);

        // No. of
        // multiplier bit
        qrn = 4;

        // sequence
        // counter
        sc = qrn;

        // multiplier
        int []qr = new int[]{ 1, 0, 1, 0 };
        Array.Reverse(qr);

        boothAlgorithm(br, qr,
                       mt, qrn, sc);

        Console.WriteLine();
        Console.Write("Result = ");

        for (int i = qrn - 1; i >= 0; i--)
            Console.Write(qr[i]);
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output :

```
qn    q[n + 1]    BR        AC    QR        sc
            initial        0000    1010        4
0    0        rightShift    0000    0101        3
1    0        A = A - BR    1010
            rightShift    1101    0010        2
0    1        A = A + BR    0011
            rightShift    0001    1001        1
1    0        A = A - BR    1011
```

```
            rightShift    1101    1100        0

Result = 1100
```

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/booths-multiplication-algorithm/

Chapter 26

Builtin functions of GCC compiler

Builtin functions of GCC compiler - GeeksforGeeks

These are four important built-in functions in GCC compiler:

1. **__builtin_popcount(x):** This function is used to count the number of one's(set bits) in an integer.
   Example:

```
if x = 4
binary value of 4 is 100
Output: No of ones is 1.

 // C program to illustrate _builtin_popcount(x)

#include <stdio.h>
int main()
{
    int n = 5;
    printf("Count of 1s in binary of %d is %d ",
           n, __builtin_popcount(n));
    return 0;
}
```

   Output:

```
Count of 1s in binary of 5 is 2
```

2. **__builtin_parity(x):** This function is used to check the parity of a number. This function returns true(1) if the number has odd parity else it returns false(0) for even parity.
Example:

```
if x = 7
7 has odd no. of 1's in its binary(111).
Output: Parity of 7 is 1
```

```
 // C program to illustrate _builtin_parity(x)

#include <stdio.h>
int main()
{
    int n = 7;
    printf("Parity of %d is %d ",
           n, __builtin_parity(n));
    return 0;
}
```

Output:

```
Parity of 7 is 1
```

3. **___builtin_clz(x):** This function is used to count the leading zeros of the integer. Note : clz = count leading zero's
Example: It counts number of zeros before the first occurence of one(set bit).

```
a = 16
Binary form of 16 is 00000000 00000000 00000000 00010000
Output: 27
```

```
 // C program to illustrate __builtin_clz(x)
#include <stdio.h>
int main()
{
    int n = 16;
    printf("Count of leading zeros before 1 in %d is %d",
           n, __builtin_clz(n));
    return 0;
}
```

Output:

```
Count of leading zeros before 1 in 16 is 27
```

Note: ___builtin_clz(x) This function only accept unsigned values

4. **___builtin_ctz(x):** This function is used to count the trailing zeros of the given integer. Note : ctz = count trailing zeros.
Example: Count no of zeros from last to first occurence of one(set bit).

```
a = 16
Binay form of 16 is 00000000 00000000 00000000 00010000
Output: ctz = 4
```

```
 // C program to illustrate __builtin_ctz(x)
#include <stdio.h>
int main()
{
    int n = 16;
    printf("Count of zeros from last to first "
           "occurrence of one is %d",
           __builtin_ctz(n));
    return 0;
}
```

Output:

```
Count of zeros from last to first occurrence of one is 4
```

Source

https://www.geeksforgeeks.org/builtin-functions-gcc-compiler/

Chapter 27

C++ bitset and its application

C++ bitset and its application - GeeksforGeeks

A bitset is an array of bool but each Boolean value is not stored separately instead bitset optimizes the space such that each bool takes 1 bit space only, so **space taken by bitset bs is less than that of bool bs[N] and vector bs(N)**. However, a limitation of bitset is, **N must be known at compile time, i.e., a constant** (this limitation is not there with vector and dynamic array)

As bitset stores the same information in compressed manner the operation on bitset are faster than that of array and vector. We can access each bit of bitset individually with help of array indexing operator [] that is bs[3] shows bit at index 3 of bitset bs just like a simple array. Remember bitset starts its indexing backward that is for 10110, 0 are at 0th and 3rd indices whereas 1 are at 1st 2nd and 4th indices.
We can construct a bitset using integer number as well as binary string via constructors which is shown in below code. The size of bitset is fixed at compile time that is, it can't be changed at runtime.
The **main function defined for bitset class** are operator [], count, size, set, reset and many more they are explained in below code –

```
 // C++ program to demonstrate various functionality of bitset
#include <bits/stdc++.h>
using namespace std;

#define M 32

int main()
{
    // default constructor initializes with all bits 0
    bitset<M> bset1;

    // bset2 is initialized with bits of 20
    bitset<M> bset2(20);
```

```
// bset3 is initialized with bits of specified binary string
bitset<M> bset3(string("1100"));

// cout prints exact bits representation of bitset
cout << bset1 << endl; // 00000000000000000000000000000000
cout << bset2 << endl; // 00000000000000000000000000010100
cout << bset3 << endl; // 00000000000000000000000000001100
cout << endl;

// declaring set8 with capacity of 8 bits

bitset<8> set8; // 00000000

// setting first bit (or 6th index)
set8[1] = 1; // 00000010
set8[4] = set8[1]; // 00010010
cout << set8 << endl;

// count function returns number of set bits in bitset
int numberof1 = set8.count();

// size function returns total number of bits in bitset
// so there difference will give us number of unset(0)
// bits in bitset
int numberof0 = set8.size() - numberof1;

cout << set8 << " has " << numberof1 << " ones and "
     << numberof0 << " zeros\n";

// test function return 1 if bit is set else returns 0
cout << "bool representation of " << set8 << " : ";
for (int i = 0; i < set8.size(); i++)
    cout << set8.test(i) << " ";

cout << endl;

// any function returns true, if atleast 1 bit
// is set
if (!set8.any())
    cout << "set8 has no bit set.\n";

if (!bset1.any())
    cout << "bset1 has no bit set.\n";

// none function returns true, if none of the bit
// is set
if (!bset1.none())
    cout << "bset1 has some bit set\n";
```

```
    // bset.set() sets all bits
    cout << set8.set() << endl;

    // bset.set(pos, b) makes bset[pos] = b
    cout << set8.set(4, 0) << endl;

    // bset.set(pos) makes bset[pos] = 1  i.e. default
    // is 1
    cout << set8.set(4) << endl;

    // reset function makes all bits 0
    cout << set8.reset(2) << endl;
    cout << set8.reset() << endl;

    // flip function flips all bits i.e.  1 <-> 0
    // and  0 <-> 1
    cout << set8.flip(2) << endl;
    cout << set8.flip() << endl;

    // Converting decimal number to binary by using bitset
    int num = 100;
    cout << "\nDecimal number: " << num
         << "  Binary equivalent: " << bitset<8>(num);

    return 0;
}
```

Output :

```
00000000000000000000000000000000
00000000000000000000000000010100
00000000000000000000000000001100

00010010
00010010 has 2 ones and 6 zeros
bool representation of 00010010 : 0 1 0 0 1 0 0 0
bset1 has no bit set.
11111111
11101111
11111111
11111011
00000000
00000100
11111011

Decimal number: 100 Binary equivalent: 01100100
```

For bitset set, reset and flip function are defined. Set function sets (1) all bits of bitset if no argument is provided otherwise it sets the bit whose position is given as argument. In same way reset and flip also work if they are called with no argument they perform their operation on whole bitset and if some position is provided as argument then they perform operation at that position only.
For bitset all bitwise operator are overloaded that is they can be applied to bitset directly without any casting or conversion, main overloaded operator are &, |, ==, != and shifting operator <> which makes operation on bitset easy.
Use of above operator is shown in below code.

```
 // C++ program to show applicable operator on bitset.
#include <bits/stdc++.h>
using namespace std;

int main()
{
    bitset<4> bset1(9); // bset1 contains 1001
    bitset<4> bset2(3); // bset2 contains 0011

    // comparison operator
    cout << (bset1 == bset2) << endl; // false 0
    cout << (bset1 != bset2) << endl; // true  1

    // bitwise operation and assignment
    cout << (bset1 ^= bset2) << endl; // 1010
    cout << (bset1 &= bset2) << endl; // 0010
    cout << (bset1 |= bset2) << endl; // 0011

    // left and right shifting
    cout << (bset1 <<= 2) << endl; // 1100
    cout << (bset1 >>= 1) << endl; // 0110

    // not operator
    cout << (~bset2) << endl; // 1100

    // bitwise operator
    cout << (bset1 & bset2) << endl; // 0010
    cout << (bset1 | bset2) << endl; // 0111
    cout << (bset1 ^ bset2) << endl; // 0101
}
```

Output :

```
0
1
1010
0010
```

```
0011
1100
0110
1100
0010
0111
0101
```

This article is contributed by Utkarsh Trivedi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : gaurav1614

Source

https://www.geeksforgeeks.org/c-bitset-and-its-application/

Chapter 28

C++ bitset interesting facts

C++ bitset interesting facts - GeeksforGeeks

Bitset is a container in C++ Standard Template Library for dealing with data at the bit level.

A bitset stores bits (elements with only two possible values: 0 or 1). We can however get the part of a string by providing positions to bitset constructor (Positions are with respect to string position from left to right)

Example:

```
 // C++ program to demonstrate that we can get part of a
// bit string in bitset.
#include <bitset>
#include <string>
#include <iostream>

int main()
{
  std::string bit_string = "110010";
  std::bitset<8> b1(bit_string);             // [0, 0, 1, 1, 0, 0, 1, 0]

  // string from position 2 till end
  std::bitset<8> b2(bit_string, 2);      // [0, 0, 0, 0, 0, 0, 1, 0]

  // string from position 2 till next 3 positions
  std::bitset<8> b3(bit_string, 2, 3);   // [0, 0, 0, 0, 0, 0, 0, 1]

  std::cout << b1 << '\n' << b2 << '\n' << b3 << '\n';

  return 0;
}
```

Output:

```
00110010
00000010
00000001
```

We can construct a bitset using the characters in the std::basic_string _str. An optional starting position _pos and length _n can be provided, as well as characters denoting alternate values for set (_one) and unset (_zero) bits.
Syntax:

```
std::bitset b1(str, pos, n, zero, one);
str    : string used to initialize the bitset
pos    : a starting offset into str
n    : number of characters to use from str
zero    : alternate character for unset bits in str
one    : alternate chracters for set bits in str
```

- If _pos > str.size(), this constructor throws std::out_of_range.
- If any characters examined in _str is not zero or one, it throws std::invalid_argument.

```
 // C++ program to demonstrate that we can construct bitset using
// alternate characters for set and unset bits.
#include <bitset>
#include <string>
#include <iostream>

int main()
{
    // string constructor using custom zero/one digits
    std::string alpha_bit_string = "aBaaBBaB";
    std::bitset<8> b1(alpha_bit_string, 0, alpha_bit_string.size(),
                      'a', 'B');         // [0,1,0,0,1,1,0,1]

    std::cout << b1 << '\n';
}
```

Output:

```
01001101
```

Constructs an object of class bitset, initializing the N bits to values that correspond to the characters provided in a c-style character string of zeros and ones. You call the constructor without casting the string into a string type. It also has two optional parameters, _Zero and _One, which indicate what character in _Str is to be interpreted to mean a 0 bit and a 1 bit, respectively.

```
 #include <bitset>
#include <iostream>

int main()
{
    // char* constructor using custom digits
    std::bitset<8> b1("XXXXYYYY", 8, 'X', 'Y'); // [0, 0, 0, 0, 1, 1, 1, 1]
    std::cout << b1 << '\n';
}
```

Output:

```
00001111
```

Bitset Operations

1. **std::bitset::to_string()**
   Converts the contents of the bitset to a string. Uses zero to represent bits with value of false and one to represent bits with value of true. The resulting string contains N characters with the first character corresponds to the last (N-1th) bit and the last character corresponding to the first bit. Also, we can pass the characters used to print true and false value through the parameters.

   Example:

   ```
    // C++ program to demonstrate that we can convert contents
   // of bitset to a string.
   #include <iostream>
   #include <bitset>

   int main()
   {
       std::bitset<8> b(42);
       std::cout << b.to_string() << '\n'
                 << b.to_string('*') << '\n'
                 << b.to_string('O', 'X') << '\n';
   }
   ```

   Output:

   ```
   00101010
   **1*1*1*
   OOXOXOXO
   ```

2. std::bitset::to_ulong():
   Converts the contents of the bitset to an unsigned long integer. The first bit of the bitset corresponds to the least significant digit of the number and the last bit corresponds to the most significant digit. Function throws std::overflow_error if the value cannot be represented in unsigned long.

   Example:

```
 // C++ program to demonstrate that we can get value of bitset
// as  unsigned long integer.
#include <iostream>
#include <bitset>

int main()
{
    std::bitset<5> b(5);
    std::cout << b.to_ulong() << '\n';
}
```

   Output:

```
5
```

Source

https://www.geeksforgeeks.org/c-bitset-interesting-facts/

Chapter 29

CHAR_BIT in C

CHAR_BIT in C - GeeksforGeeks

CHAR_BIT : It is the number of bits in char. These days, almost all architectures use 8 bits per byte (But it is not the case always, some older machines used to have 7-bit byte). It can be found in

Let us see an application of it. Suppose we wish to print byte by byte representation of an integer.

Examples :

```
Input  : 4
Output : 00000000 00000000 00000000 00000100

Input  : 12
Output : 00000000 00000000 00000000 00001100
```

```
 // CPP program to print byte by byte presentation
#include <bits/stdc++.h>
using namespace std;

// function in which number and intitally 0 is passed
void printInBinary(int num)
{
    int n = CHAR_BIT*sizeof(num);
    stack<bool> s;
    for (int i=1; i<=n; i++)
    {
        s.push(num%2);
        num = num/2;
    }
    for (int i=1; i<=n; i++)
```

```
    {
        cout << s.top();
        s.pop();

        // Put a space after every byte.
        if (i % CHAR_BIT == 0)
        cout << " ";
    }
}

int main()
{
    int num = 12;
    printInBinary(num);
    return 0;
}
```

Output :

```
00000000 00000000 00000000 00001100
```

Source

https://www.geeksforgeeks.org/char_bit-in-c/

Chapter 30

Calculate 7n/8 without using division and multiplication operators

Calculate 7n/8 without using division and multiplication operators - GeeksforGeeks

Given an integer, write a function that calculates 7n/8 (ceilingof 7n/8) without using division and multiplication operators.

We strongly recommend to minimize your browser and try this yourself first.

Method 1:
The idea is to first calculate floor of n/8, i.e., n/8 using right shift bitwise operator. The expression n>>3 produces the same.

If we subtract n/8 from n, we get 7n/8

Below is the implementation of above idea :

CPP

```
 // C program to evaluate ceil(7n/8) without using * and /
#include<stdio.h>

int multiplyBySevenByEight(unsigned int n)
{
    /* Note the inner bracket here. This is needed
       because precedence of '-' operator is higher
       than '<<' */
    return (n - (n>>3));
}

/* Driver program to test above function */
```

```
int main()
{
    unsigned int n = 9;
    printf("%d", multiplyBySevenByEight(n));
    return 0;
}
```

Java

```
 // Java program to evaluate ceil(7n/8)
// without using * and
import java.io.*;

class GFG
{
    static int multiplyBySevenByEight(int n)
    {
        /* Note the inner bracket here. This is needed
        because precedence of '-' operator is higher
        than '<<' */
        return (n - (n >> 3));
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 9;
        System.out.println(multiplyBySevenByEight(n));
    }
}

//This code is contributed by Anshika Goyal.
```

Python3

```
 # Python program to evaluate ceil(7n/8) without using * and /

def multiplyBySevenByEight(n):

    # Note the inner bracket here. This is needed
    # because precedence of '-' operator is higher
    # than '<<'
    return (n - (n>>3))


# Driver program to test above function */
n = 9
```

```
print(multiplyBySevenByEight(n))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to evaluate ceil(7n/8)
// without using * and
using System;

public class GFG {

    static int multiplyBySevenByEight(int n)
    {
        /* Note the inner bracket here.
        This is needed because precedence
        of '-' operator is higher than
        '<<' */
        return (n - (n >> 3));
    }

    // Driver code
    public static void Main()
    {
        int n = 9;

        Console.WriteLine(
            multiplyBySevenByEight(n));
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to evaluate ceil
// (7n/8) without using * and

function multiplyBySevenByEight( $n)
{
    // Note the inner bracket here.
    // This is needed because
    // precedence of '-' operator
    // is higher than '<<'
```

```
    return ($n - ($n >> 3));
}

// Driver Code
$n = 9;
echo multiplyBySevenByEight($n);

// This code is contributed by Ajit
?>
```

Output :

```
8
```

Method 2 (Always matches with 7*n/8):
The above method doesn't always produce result same as "printf("%u", 7*n/8)". For example, the value of expression 7*n/8 is 13 for n = 15, but above proogram produces 14. Below is modified version that always matches 7*n/8. The idea is to first multiply the number with 7, then divide by 8 as it happens in expression 7*n/8.

C++

```
 // C program to evaluate 7n/8 without using * and /
#include<stdio.h>

int multiplyBySevenByEight(unsigned int n)
{
    /* Step 1) First multiply number by 7 i.e. 7n = (n << 3) -n
     * Step 2) Divide result by 8 */
   return ((n << 3) -n) >> 3;
}

/* Driver program to test above function */
int main()
{
    unsigned int n = 15;
    printf("%u", multiplyBySevenByEight(n));
    return 0;
}
```

Java

```
 // Java program to evaluate 7n/8
// without using * and /
import java.io.*;
```

```
class GFG
{

    static int multiplyBySevenByEight(int n)
    {
        // Step 1) First multiply number
        // by 7 i.e. 7n = (n << 3) -n
        // * Step 2) Divide result by 8
        return ((n << 3) -n) >> 3;
    }

    // Driver program
    public static void main(String args[])
    {

        int n = 15;
        System.out.println(multiplyBySevenByEight(n));
    }
}

// This code is contributed by Anshika Goyal.
```

Python3

```
 # python program to evaluate 7n/8
# without using * and /

def multiplyBySevenByEight(n):

    #Step 1) First multiply number
    # by 7 i.e. 7n = (n << 3) -n
    # Step 2) Divide result by 8
    return ((n << 3) -n) >> 3;

# Driver code
n = 15;
print(multiplyBySevenByEight(n));

#this code is contributed by sam007.
```

C#

```
 // C# program to evaluate 7n/8
// without using * and /
using System;

public class GFG {
```

```
    static int multiplyBySevenByEight(int n)
    {

        // Step 1) First multiply number
        // by 7 i.e. 7n = (n << 3) -n
        // * Step 2) Divide result by 8
        return ((n << 3) -n) >> 3;
    }

    // Driver program
    public static void Main()
    {
        int n = 15;

        Console.WriteLine(
            multiplyBySevenByEight(n));
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to evaluate 7n/8
// without using * and /

function multiplyBySevenByEight( $n)
{

     /* Step 1) First multiply number by
                7 i.e. 7n = (n << 3) - n
         Step 2) Divide result by 8 */
    return (($n << 3) -$n) >> 3;
}

    // Driver Code
    $n = 15;
    echo multiplyBySevenByEight($n);

// This code is contributed by anuj_67.
?>
```

Output :

```
13
```

Note : There is difference between outcomes of two methods. The method 1 produces ceil(7n/8), but method two produces integer value of 7n/8. For example, for n = 15, outcome of first method is 14, but for second method is 13.

Thanks to **Narendra Kangralkar** for suggesting this method.

This article is contributed by **Rajeev**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : Sam007, jit_t, vt_m

Source

https://www.geeksforgeeks.org/calculate-7n8-without-using-division-and-multiplication-operators/

Chapter 31

Calculate XOR from 1 to n.

Calculate XOR from 1 to n. - GeeksforGeeks

Given a number n, the task is to find the XOR from 1 to n.

Examples :

```
Input : n = 6
Output : 7
// 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6  = 7

Input : n = 7
Output : 0
// 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 ^ 7 = 0
```

Method 1 (Naive Approach):
1- Initialize result as 0.
1- Traverse all numbers from 1 to n.
2- Do XOR of numbers one by one with result.
3- At the end, return result.

Method 2 (Efficient method) :
1- Find the remainder of n by moduling it with 4.
2- If rem = 0, then xor will be same as n.
3- If rem = 1, then xor will be 1.
4- If rem = 2, then xor will be n+1.
5- If rem = 3 ,then xor will be 0.

C/C++

```
 // C++ program to find XOR of numbers
// from 1 to n.
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to calculate xor
long computeXOR(const int n)
{
    // Modulus operator are expensive on most of the
    // computers. n & 3 will be equivalent to n % 4.

    switch(n & 3) // n % 4
    {
    case 0: return n;     // if n is multiple of 4
    case 1: return 1;     // If n % 4 gives remainder 1
    case 2: return n + 1; // If n % 4 gives remainder 2
    case 3: return 0;     // If n % 4 gives remainder 3
    }
}

// Driver code
int main()
{
    // your code goes here
    int n = 5;
    cout <<    computeXOR(n);
    return 0;
}
```

Java

```
 // Java program to find XOR of numbers
// from 1 to n.

class GFG
{
    // Method to calculate xor
    static int computeXOR(int n)
    {
        // If n is a multiple of 4
        if (n % 4 == 0)
            return n;

        // If n%4 gives remainder 1
        if (n % 4 == 1)
            return 1;

        // If n%4 gives remainder 2
        if (n % 4 == 2)
            return n + 1;
```

```
        // If n%4 gives remainder 3
        return 0;
    }

    // Driver method
    public static void main (String[] args)
    {
         int n = 5;
         System.out.println(computeXOR(n));
    }
}
```

Python 3

```
 # Python 3 Program to find
# XOR of numbers from 1 to n.

# Function to calculate xor
def computeXOR(n) :

    # Modulus operator are expensive
    # on most of the computers. n & 3
    # will be equivalent to n % 4.

    # if n is multiple of 4
    if n % 4 == 0 :
        return n

    # If n % 4 gives remainder 1
    if n % 4 == 1 :
        return 1

    # If n%4 gives remainder 2
    if n % 4 == 2 :
        return n + 1

    # If n%4 gives remainder 3
    return 0

# Driver Code
if __name__ == "__main__" :

    n = 5

    # function calling
    print(computeXOR(n))

# This code is contributed by ANKITRAI1
```

C#

```
 // C# program to find XOR
// of numbers from 1 to n.
using System;

class GFG
{

    // Method to calculate xor
    static int computeXOR(int n)
    {
        // If n is a multiple of 4
        if (n % 4 == 0)
            return n;

        // If n%4 gives remainder 1
        if (n % 4 == 1)
            return 1;

        // If n%4 gives remainder 2
        if (n % 4 == 2)
            return n + 1;

        // If n%4 gives remainder 3
        return 0;
    }

    // Driver Code
    static public void Main ()
    {
        int n = 5;
        Console.WriteLine(computeXOR(n));
    }
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP program to find XOR
// of numbers from 1 to n.

// Function to calculate xor
function computeXOR($n)
{
```

```
    // Modulus operator are expensive
    // on most of the computers. n & 3
    // will be equivalent to n % 4.

    switch($n & 3) // n % 4
    {
    // if n is multiple of 4
    case 0: return $n;

    // If n % 4 gives remainder 1
    case 1: return 1;

    // If n % 4 gives remainder 2
    case 2: return $n + 1;

    // If n % 4 gives remainder 3
    case 3: return 0;
    }
}

// Driver code
$n = 5;
echo computeXOR($n);

// This code is contributed by aj_36
?>
```

Output :

```
1
```

How does this work?
When we do XOR of numbers, we get 0 as XOR value just before a multiple of 4. This keeps repeating before every multiple of 4.

```
Number Binary-Repr  XOR-from-1-to-n
1         1           [0001]
2        10           [0011]
3        11           [0000]  <----- We get a 0
4       100           [0100]  <----- Equals to n
5       101           [0001]
6       110           [0111]
7       111           [0000]  <----- We get 0
8      1000           [1000]  <----- Equals to n
9      1001           [0001]
```

```
10      1010            [1011]
11      1011            [0000] <------ We get 0
12      1100            [1100] <------ Equals to n
```

Improved By : Sameer Chaudhari 1, jit_t, ANKITRAI1

Source

https://www.geeksforgeeks.org/calculate-xor-1-n/

Chapter 32

Calculate square of a number without using *, / and pow()

Calculate square of a number without using *, / and pow() - GeeksforGeeks

Given an integer n, calculate square of a number without using *, / and pow().

Examples :

```
Input: n = 5
Output: 25

Input: 7
Output: 49

Input: n = 12
Output: 144
```

A **Simple Solution** is to repeatedly add n to result. Below is the implementation of this idea.

C++

```
 // Simple solution to calculate square without
// using * and pow()
#include<iostream>
using namespace std;

int square(int n)
{
   // handle negative input
   if (n<0) n = -n;
```

```
   // Initialize result
   int res = n;

   // Add n to res n-1 times
   for (int i=1; i<n; i++)
       res += n;

   return res;
}

// drive program
int main()
{
    for (int n = 1; n<=5; n++)
        cout << "n = " << n << ", n^2 = "
             << square(n) << endl;
    return 0;
}
```

Java

```
 // Java Simple solution to calculate
// square without using * and pow()
import java.io.*;

class GFG {

    public static int square(int n) {

        // handle negative input
        if (n < 0)
            n = -n;

        // Initialize result
        int res = n;

        // Add n to res n-1 times
        for (int i = 1; i < n; i++)
            res += n;

        return res;
    }

    // Driver code
    public static void main(String[] args) {

        for (int n = 1; n <= 5; n++)
```

```
            System.out.println("n = " + n +
                   ", n^2 = " + square(n));
    }
}

// This code is contributed by sunnysingh
```

Python3

```
 # Simple solution to
# calculate square without
# using * and pow()

def square(n):

    # handle negative input
    if (n < 0):
        n = -n

    # Initialize result
    res = n

    # Add n to res n-1 times
    for i in range(1, n):
        res += n

    return res


# Driver Code
for n in range(1, 6):
    print("n =", n , end=", ")
    print("n^2 =", square(n))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# Simple solution to calculate
// square without using * and pow()
using System;

class GFG
{
    public static int square(int n) {
```

```
        // handle negative input
        if (n < 0)
            n = -n;

        // Initialize result
        int res = n;

        // Add n to res n-1 times
        for (int i = 1; i < n; i++)
            res += n;

        return res;
    }

    // Driver code
    public static void Main() {

        for (int n = 1; n <= 5; n++)
        Console.WriteLine("n = " + n +
                ", n^2 = " + square(n));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP implementation to
// calculate square
// without using * and pow()

function square($n)
{

// handle negative input
    if ($n < 0) $n = -$n;

// Initialize result
        $res = $n;

// Add n to res n-1 times
for ($i = 1; $i < $n; $i++)
    $res += $n;

return $res;
}
```

```
// Drive Code
for ($n = 1; $n<=5; $n++)
    echo "n = ", $n, ", ", "n^2 = ",
                  square($n), "\n ";

// This code is contributed by Ajit
?>
```

Output :

```
n = 1, n^2 = 1
n = 2, n^2 = 4
n = 3, n^2 = 9
n = 4, n^2 = 16
n = 5, n^2 = 25
```

Time complexity of above solution is O(n). We can do it in **O(Logn) time using bitwise operators**. The idea is based on the following fact.

```
  square(n) = 0 if n == 0
  if n is even
     square(n) = 4*square(n/2)
  if n is odd
     square(n) = 4*square(floor(n/2)) + 4*floor(n/2) + 1

Examples
  square(6) = 4*square(3)
  square(3) = 4*(square(1)) + 4*1 + 1 = 9
  square(7) = 4*square(3) + 4*3 + 1 = 4*9 + 4*3 + 1 = 49
```

How does this work?

```
If n is even, it can be written as
  n = 2*x
  n2 = (2*x)2 = 4*x2
If n is odd, it can be written as
  n = 2*x + 1
  n2 = (2*x + 1)2 = 4*x2 + 4*x + 1
```

floor(n/2) can be calculated using bitwise right shift operator. 2*x and 4*x can be calculated

Below is the implementation based on above idea.

C++

```
 // Square of a number using bitwise operators
#include<iostream>
using namespace std;

int square(int n)
{
    // Base case
    if (n==0) return 0;

    // Handle negative number
    if (n < 0) n = -n;

    // Get floor(n/2) using right shift
    int x = n>>1;

    // If n is odd
    if (n&1)
        return ((square(x)<<2) + (x<<2) + 1);
    else // If n is even
        return (square(x)<<2);
}

// Driver Code
int main()
{
    for (int n = 1; n<=5; n++)
        cout << "n = " << n << ", n^2 = " << square(n) << endl;
    return 0;
}
```

Java

```
 // Square of a number using
// bitwise operators
class GFG
{
    static int square(int n)
    {

        // Base case
        if (n == 0)
            return 0;

        // Handle negative number
        if (n < 0)
            n = -n;

        // Get floor(n/2) using
```

```
        // right shift
        int x = n >> 1;

        // If n is odd
        ;
        if (n % 2 != 0)
            return ((square(x) << 2)
                    + (x << 2) + 1);
        else // If n is even
            return (square(x) << 2);
    }



    public static void main(String args[])
    {
        for (int n = 1; n <= 5; n++)
        System.out.println("n = " + n +
                           " n^2 = " +
                           square(n));
    }
}

// This code is contributed by Sam007
```

Python3

```
 # Square of a number using bitwise
# operators

def square(n):

    # Base case
    if (n == 0):
        return 0

    # Handle negative number
    if (n < 0):
        n = -n

    # Get floor(n/2) using
    # right shift
    x = n>>1

    # If n is odd
    if (n & 1):
        return ((square(x) << 2)
                  + (x << 2) + 1);
```

```
    # If n is even
    else:
        return (square(x) << 2);

# Driver Code
for n in range (1, 6):
    print("n = " , n ," n^2 = ",
                        square(n))
# This code is contributed by Sam007
```

C#

```
 // Square of a number using bitwise
// operators
using System;

class GFG {

    static int square(int n)
    {

        // Base case
        if (n == 0)
            return 0;

        // Handle negative number
        if (n < 0)
            n = -n;

        // Get floor(n/2) using
        // right shift
        int x = n>>1;

        // If n is odd
        ;
        if (n % 2 != 0)
            return ((square(x) << 2)
                     + (x << 2) + 1);
        else // If n is even
            return (square(x)<<2);
    }

    static void Main()
    {
        for (int n = 1; n <= 5; n++)
        Console.WriteLine( "n = " + n
            + " n^2 = " + square(n));
```

```
    }
}

// This code is contributed by Sam0007.
```

PHP

```
 <?php
// Square of a number using
// bitwise operators

function square($n)
{

    // Base case
    if ($n==0) return 0;

    // Handle negative number
    if ($n < 0) $n = -$n;

    // Get floor(n/2)
    // using right shift
    $x = $n >> 1;

    // If n is odd
    if ($n & 1)
        return ((square($x) << 2) +
                    ($x << 2) + 1);
    else // If n is even
        return (square($x) << 2);
}

    // Driver Code
    for ($n = 1; $n <= 5; $n++)
        echo "n = ", $n, ", n^2 = ", square($n),"\n";

// This code is contributed by ajit
?>
```

Output :

```
n = 1, n^2 = 1
n = 2, n^2 = 4
n = 3, n^2 = 9
n = 4, n^2 = 16
n = 5, n^2 = 25
```

Time complexity of the above solution is O(Logn).

This article is contributed by **Ujjwal Jain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : Sam007, jit_t

Source

https://www.geeksforgeeks.org/calculate-square-of-a-number-without-using-and-pow/

Chapter 33

Calculate the total fine to be collected

Calculate the total fine to be collected - GeeksforGeeks

Given a date and an array of integer containing the numbers of the cars traveling on that date(an integer), the task is to calculate the total fine collected based on the following rules:

- Odd numbered cars can travel on only odd dates.
- Even numbered cars on only even dates.
- Otherwise a car would be fined 250 Rs.

Examples:

```
Input: car_num[] = {3, 4, 1, 2}, date = 15
Output: 500
Car with numbers '4' and '2' will be fined
250 each.

Input: car_num[] = {1, 2, 3} , date = 16
Output: 500
Car with numbers '1' and '3' will be fined
250 each.
```

Approach:

1. Start traversing the given array.
2. Check if the current car number and date doesn't match i.e. one is even and other is odd or vice-versa.
3. If not matched charge the fine on that car number. Else, not.

4. Print the total fine.

Below is the implementation of above approach:

C++

```
 // C++ implementation to calculate
// the total fine collected
#include <bits/stdc++.h>
using namespace std;

// function to calculate the total fine collected
int totFine(int car_num[], int n, int date, int fine)
{
    int tot_fine = 0;

    // traverse the array elements
    for (int i = 0; i < n; i++)

        // if both car no and date are odd or
        // both are even, then statement
        // evaluates to true
        if (((car_num[i] ^ date) & 1) == 1)
            tot_fine += fine;

    // required total fine
    return tot_fine;
}

// Driver program to test above
int main()
{
    int car_num[] = { 3, 4, 1, 2 };
    int n = sizeof(car_num) / sizeof(car_num[0]);
    int date = 15, fine = 250;

    cout << totFine(car_num, n, date, fine);

    return 0;
}
```

Java

```
// Java implementation to calculate
// the total fine collected
class GFG
{
```

// function to calculate
// the total fine collected
static int totFine(int car_num[], int n,
int date, int fine)
{
int tot_fine = 0;

// traverse the array elements
for (int i = 0; i < n; i++) // if both car no and date // are odd or both are even, // then statement evaluates to true if (((car_num[i] ^ date) & 1) == 1) tot_fine += fine; // required total fine return tot_fine; } // Driver Code public static void main(String[] args) { int car_num[] = { 3, 4, 1, 2 }; int n = car_num.length; int date = 15, fine = 250; System.out.println(totFine(car_num, n, date, fine)); } } // This code is contributed // by ChitraNayal [tabby title = "Python 3"]

```
 # Python 3 program to calculate
# the total fine collected

# function to calculate the total fine collected
def totFine(car_num, n, date, fine) :

    tot_fine = 0

    # traverse the array elements
    for i in range(n) :

        # if both car no and date are odd or
        # both are even, then statement
        # evaluates to true
        if (((car_num[i] ^ date) & 1) == 1 ):
            tot_fine += fine

    # required total fine
    return tot_fine

# Driver Program
if __name__ == "__main__" :

    car_num = [ 3, 4, 1, 2 ]
    n = len(car_num)
    date, fine = 15, 250

    # function calling
    print(totFine(car_num, n, date, fine))

# This code is contributed by ANKITRAI1
```

C#

```
// C# implementation to calculate
// the total fine collected
using System;

class GFG
{

// function to calculate the
// total fine collected
static int totFine(int[] car_num, int n,
int date, int fine)
{
int tot_fine = 0;

// traverse the array elements
for (int i = 0; i < n; i++) // if both car no and date // are odd or both are even, // then statement evaluates to true if (((car_num[i] ^ date) & 1) == 1) tot_fine += fine; // required total fine return tot_fine; } // Driver Code public static void Main() { int[] car_num = { 3, 4, 1, 2 }; int n = car_num.Length; int date = 15, fine = 250; Console.Write(totFine(car_num, n, date, fine)); } } // This code is contributed // by ChitraNayal [tabby title="PHP"]
```

Output:

```
500
```

Source:https://www.geeksforgeeks.org/microsoft-interview-experience-for-internship/

Improved By : ANKITRAI1, ChitraNayal

Source

https://www.geeksforgeeks.org/calculate-the-total-fine-to-be-collected/

Chapter 34

Case conversion (Lower to Upper and Vice Versa) of a string using BitWise operators in C/C++

Case conversion (Lower to Upper and Vice Versa) of a string using BitWise operators in C/C++ - GeeksforGeeks

Given a string, write a function that converts it either from lower to upper case or from upper to lower case using the bitwise operators &(AND), |(OR), ~(NOT) in place and returns the string.

Many of us know that Bitwise manipulations are faster than performing arithmetic operations for a compiler as the data is stored in binary form 0's and 1's.

Examples:

```
Input : "LowerToUpPer"
Output : "LOWERTOUPPER"
   Letters already in the uppercase remains the same.
   while rest get converted to uppercase.

Input : "UPPerTOloweR"
Output : "uppertolower"
   Letters already in the lowercase remains the same.
   while rest get converted to lowercase.
```

1.Lower to Upper Case
This method simply subtracts a value of 32 from the ASCII value of lowercase letter by Bitwise ANDing (&) with negation (~) of 32 converting the letter to uppercase.

```
 // C++ program to convert a string from
// lower to upper case.
#include<stdio.h>

const int x = 32;

// Converts a string to uppercase
char *toUpperCase(char *a)
{
    for (int i=0; a[i]!='\0'; i++)
        a[i] = a[i] & ~x;

    return a;
}

// Driver Code
int main()
{
    char str[] = "SanjaYKannA";

    //Here it's recommended to use character array
    //as it's stored in read-write area.
    //If a pointer is used it's stored
    //in read-only memory as a string literal.

    printf("%s", toUpperCase(str));

    return 0;
}
```

Output:

```
SANJAYKANNA
```

2.Upper to Lower Case
Similarly, it adds a value of 32 to the ASCII value of uppercase letter by Bitwise ORing (|) with 32 converting the letter to lowercase.

```
 // C++ program to convert a string from
// upper to lower case.
#include<stdio.h>
const int x = 32;

// Converts a string to lowercase
char * toLowerCase(char *a)
{
```

```
    for (int i=0; a[i]!='\0'; i++)
        a[i] = a[i] | x;

    return a;
}

// Driver Code
int main()
{
    char str[] = "SanjaYKannA";
    printf("%s", toLowerCase(str));
    return 0;
}
```

Output:

```
sanjaykanna
```

Explanation:
TheASCII table is constructed in such way that the binary representation of lowercase letters is almost identical of binary representation of uppercase letters.

Character 'A' is integer 65 = (0100 0001)2, while character 'a' is integer 97 = (0110 0001)2. The difference between the ASCII values of 'a' and 'A' is 32.

So we can easily change the case of the letters either from Upper to lower or lower to upper by adding or subtracting the difference from the letters using bitwise operators as shown above.

Exercise:
Implement a function that change the case of a string such that GeeksFoRgeekS turns gEEKSfOrGEEKs .

Source

https://www.geeksforgeeks.org/case-conversion-lower-upper-vice-versa-string-using-bitwise-operators-cc/

Chapter 35

Change all even bits in a number to 0

Change all even bits in a number to 0 - GeeksforGeeks

Given a number, change all bits at even positions to 0.

Examples:

```
Input : 30
Output : 10
Binary representation of 11110.
Bits at Even positions are highlighted.
After making all of them 0, we get 01010

Input :  10
Output :  10
```

Method 1 (Bit Traversal)

The idea is to traverse through all even bits. We accumulate all powers of 2 in a number to subtract. Finally we subtract the accumulated value from n to obtain the result.

C++

```
 // C++ program to change even
// bits to 0.
#include<bits/stdc++.h>
using namespace std;

// Returns modified number with
// all even bits 0.
```

```
int changeEvenBits(int n)
{
    // To store sum of bits
    // at even positions.
    int to_subtract = 0;

    // To store bits to shift
    int m = 0;

    // One by one put all even
    // bits to end
    for (int x = n; x; x >>= 2)
    {
        // If current last bit
        // is set, add it to ans
        if (x & 1)
           to_subtract += (1 << m);

        // Next shift position
        m += 2;
    }

    return n - to_subtract;
}

// Driver code
int main()
{
    int n = 30;
    cout << changeEvenBits(n) << endl;

    return 0;
}
```

Java

```
 // Java program to change even
// bits to 0.
import java.util.*;
class GFG
{
// Returns modified number with
// all even bits 0.
static int changeEvenBits(int n)
{
    // To store sum of bits
    // at even positions.
    int to_subtract = 0;
```

```
    // To store bits to shift
    int m = 0;

    // One by one put all even
    // bits to end
    for (int x = n; x>0; x >>= 2)
    {
        // If current last bit
        // is set, add it to ans
        if ((x & 1) > 0)
        to_subtract += (1 << m);

        // Next shift position
        m += 2;
    }

    return n - to_subtract;
}

// Driver code
    public static void main(String[] args)
    {
    int n = 30;
    System.out.println(changeEvenBits(n));
    }
}
/* This code is contributed by Mr. Somesh Awasthi */
```

Python

```
 # Python program to change even
# bits to 0.

# Returns modified number with
# all even bits 0.
def changeEvenBits(n):

    # To store sum of bits
    # at even positions.
    to_subtract = 0

    # To store bits to shift
    m = 0

    # One by one put all even
    # bits to end
    x = n
```

```
    while(x):

        # If current last bit
        # is set, add it to ans
        if (x & 1):
           to_subtract += (1 << m)

        # Next shift position
        m += 2
        x >>= 2

    return n - to_subtract

# Driver code
n = 30
print changeEvenBits(n)

# This code is contributed by Sachin Bisht
```

C#

```
 // C# program to change even
// bits to 0.
using System;

class GFG {

    // Returns modified number with
    // all even bits 0.
    static int changeEvenBits(int n)
    {

        // To store sum of bits
        // at even positions.
        int to_subtract = 0;

        // To store bits to shift
        int m = 0;

        // One by one put all even
        // bits to end
        for (int x = n; x > 0; x >>= 2)
        {

            // If current last bit
            // is set, add it to ans
            if ((x & 1) > 0)
            to_subtract += (1 << m);
```

```
            // Next shift position
            m += 2;
        }

        return n - to_subtract;
    }

    // Driver code
    public static void Main()
    {
        int n = 30;

        Console.Write(changeEvenBits(n));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to change even
// bits to 0.

// Returns modified number with
// all even bits 0.
function changeEvenBits($n)
{

    // To store sum of bits
    // at even positions.
    $to_subtract = 0;

    // To store bits to shift
    $m = 0;

    // One by one put all even
    // bits to end
    for ($x = $n; $x; $x >>= 2)
    {

        // If current last bit
        // is set, add it to ans
        if ($x & 1)
        $to_subtract += (1 << $m);

        // Next shift position
```

```
        $m += 2;
    }

    return $n - $to_subtract;
}

    // Driver code
    $n = 30;
    echo changeEvenBits($n) ;

// This code is contributed by nitin mittal
?>
```

Output :

```
10
```

Method 2 (Bitmask)

C++

```
 #include<bits/stdc++.h>
using namespace std;

int convertEvenBitToOne(int n) {
    return (n & 0xaaaaaaaa);
}

int main() {
   int n = 30;
   cout << convertEvenBitToOne(n);
   return 0;
}
```

Java

```
 // Java program using Bitmask to
// Change all even bits in a
// number to 0
import java.io.*;

class GFG {

    static int convertEvenBitToOne(int n)
```

```
    {
        return (n & 0xaaaaaaaaa);
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 30;
        System.out.println(
                    convertEvenBitToOne(n));
    }
}

// This code is contributed by anuj_67.
```

Python

```
 def convertEvenBitToOne(n):
    return (n & 0xaaaaaaaaa)

# Driver code
n = 30
print convertEvenBitToOne(n)

# This code is contributed by Sachin Bisht
```

C#

```
 // C# program using Bitmask to
// Change all even bits in a
// number to 0
using System;

class GFG {

    static long convertEvenBitToOne(int n)
    {
        return (n & 0xaaaaaaaaa);
    }

    // Driver code
    public static void Main ()
    {
        int n = 30;
        Console.WriteLine(
                    convertEvenBitToOne(n));
    }
```

```
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program using Bitmask to
// Change all even bits in a
// number to 0

function convertEvenBitToOne($n)
{
    return ($n & 0xaaaaaaaa);
}

// Driver Code
$n = 30;
echo convertEvenBitToOne($n);

// This code is contributed by anuj_67.
?>
```

Output :

```
10
```

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/change-even-bits-number-0/

Chapter 36

Change bits to make specific OR value

Change bits to make specific OR value - GeeksforGeeks

Given two positive integers A and B, we can change at most K bits in both the numbers to make OR of them equal to a given target number T. In the case of multiple solutions try to keep A as small as possible.

Examples :

```
Input : A = 175,
        B = 66,
        T = 100,
        K = 5
Output : A = 36
         B = 64
Initial bits of A = 1010 1111
Changed bits of A = 0010 0100
Initial bits of B = 0100 0010
Changed bits of B = 0100 0000
OR of changed Bits = 0110 0100
Which has decimal value equal to Target T.

Input : A = 175,
        B = 66,
        T = 100,
        K = 4
Output : Not Possible
It is not possible to get OR of
A and B as T, just by changing K bits.
```

We can solve this problem by iterating over all bits of A and B and greedily changing them that is,

- If i-th bit of Target T is 0 then set i-th bits of A and B to 0 (if not already)
- If i-th bit of Target T is 1 then we will try to set one of the bits to 1 and we will change i-th bit of B only to 1(if not already) to minimize A.

After above procedure, if **changed bits are more than K**, then it is not possible to get OR of A and B as T by changing at most K bits.
If **changed bits are less than k**, then we can further minimize the value of A by using remaining value of K for which we will loop over bits one more time and if at any time,

- i-th A bit is 1 and i-th B bit is 0 then we will make 2 changes and flip both.
- i-th A and B bits are 1 then again we will make 1 change and flip A's bit.

Total time complexity of above solution will be O(max number of bits).

C++

```
 // C++ program to change least bits to
// get desired OR value
#include <bits/stdc++.h>
using namespace std;

// Returns max of three numbers
int max(int a, int b, int c)
{
    return max(a, max(b, c));
}

// Returns count of bits in N
int bitCount(int N)
{
    int cnt = 0;
    while (N)
    {
        cnt++;
        N >>= 1;
    }
    return cnt;
}

// Returns bit at 'pos' position
bool at_position(int num, int pos)
{
    bool bit = num & (1<<pos);
    return bit;
```

```
}

//    Utility method to toggle bit at
// 'pos' position
void toggle(int &num,int pos)
{
    num ^= (1 << pos);
}

//    method returns minimum number of bit flip
// to get T as OR value of A and B
void minChangeToReachTaregetOR(int A, int B,
                               int K, int T)
{
    int maxlen = max(bitCount(A), bitCount(B),
                                  bitCount(T));

    // Loop over maximum number of bits among
    // A, B and T
    for (int i = maxlen - 1; i >= 0; i--)
    {
        bool bitA = at_position(A, i);
        bool bitB = at_position(B, i);
        bool bitT = at_position(T, i);

        // T's bit is set, try to toggle bit
        // of B, if not already
        if (bitT)
        {
            if (!bitA && !bitB)
            {
                toggle(B, i);
                K--;
            }
        }
        else
        {
            //    if A's bit is set, flip that
            if (bitA)
            {
                toggle(A, i);
                K--;
            }

            //    if B's bit is set, flip that
            if (bitB)
            {
                toggle(B, i);
```

```
                K--;
            }
        }
    }

    //    if K is less than 0 then we can make A|B == T
    if (K < 0)
    {
        cout << "Not possible\n";
        return;
    }

    // Loop over bits one more time to minimise
    // A further
    for (int i = maxlen - 1; K > 0 && i >= 0; --i)
    {
        bool bitA = at_position(A, i);
        bool bitB = at_position(B, i);
        bool bitT = at_position(T, i);

        if (bitT)
        {
            // If both bit are set, then Unset
            // A's bit to minimise it
            if (bitA && bitB)
            {
                toggle(A, i);
                K--;
            }
        }

        // If A's bit is 1 and B's bit is 0,
        // toggle both
        if (bitA && !bitB && K >= 2)
        {
            toggle(A, i);
            toggle(B, i);
            K -= 2;
        }
    }

    //    Output changed value of A and B
    cout << A << " " << B << endl;
}

// Driver code
int main()
{
```

```
    int A = 175, B = 66, K = 5, T = 100;
    minChangeToReachTaregetOR(A, B, K, T);
    return 0;
}
```

PHP

```
 <?php
// PHP program to change least
// bits to get desired OR value

// Returns max of three numbers
function maxDD($a, $b, $c)
{
    return max($a, (max($b, $c)));
}

// Returns count of bits in N
function bitCount($N)
{
    $cnt = 0;
    while ($N)
    {
        $cnt++;
        $N >>= 1;
    }
    return $cnt;
}

// Returns bit at 'pos' position
function at_position($num, $pos)
{
    $bit = $num & (1 << $pos);
    return $bit;
}

// Utility method to toggle
// bit at 'pos' position
function toggle(&$num, $pos)
{
    $num ^= (1 << $pos);
}

// method returns minimum
// number of bit flip to
// get T as OR value of A and B
function minChangeToReachTaregetOR($A, $B,
                                   $K, $T)
```

```
{
    $maxlen = max(bitCount($A),
                  bitCount($B),
                  bitCount($T));

    // Loop over maximum number
    // of bits among A, B and T
    for ( $i = $maxlen - 1; $i >= 0; $i--)
    {
        $bitA = at_position($A, $i);
        $bitB = at_position($B, $i);
        $bitT = at_position($T, $i);

        // T's bit is set, try to toggle
        // bit of B, if not already
        if ($bitT)
        {
            if (!$bitA && !$bitB)
            {
                toggle($B, $i);
                $K--;
            }
        }
        else
        {
            // if A's bit is set,
            // flip that
            if ($bitA)
            {
                toggle($A, $i);
                $K--;
            }

            // if B's bit is set,
            // flip that
            if ($bitB)
            {
                toggle($B, $i);
                $K--;
            }
        }
    }

    // if K is less than 0 then
    // we can make A|B == T
    if ($K < 0)
    {
    echo "Not possible\n";
```

```
        return;
    }

    // Loop over bits one more
    // time to minimise A further
    for ($i = $maxlen - 1;
         $K > 0 && $i >= 0; --$i)
    {
        $bitA = at_position($A, $i);
        $bitB = at_position($B, $i);
        $bitT = at_position($T, $i);

        if ($bitT)
        {
            // If both bit are set, then
            // Unset A's bit to minimise it
            if ($bitA && $bitB)
            {
                toggle($A, $i);
                $K--;
            }
        }

        // If A's bit is 1 and B's
        // bit is 0, toggle both
        if ($bitA && !$bitB && $K >= 2)
        {
            toggle($A, $i);
            toggle($B, $i);
            $K -= 2;
        }
    }

    // Output changed value
    // of A and B
    echo $A , " " , $B , "\n";
}

// Driver Code
$A = 175;
$B = 66;
$K = 5;
$T = 100;
minChangeToReachTaregetOR($A, $B, $K, $T);

// This code is contributed by ajit
?>
```

Output :

```
36 64
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/change-bits-make-specific-value/

Chapter 37

Check a number is odd or even without modulus operator

Check a number is odd or even without modulus operator - GeeksforGeeks

Given a number, check whether it is even or odd.

Examples :

```
Input: n = 11
Output: Odd

Input: n = 10
Output: Even
```

Method 1: Using Loop.
The idea is to start with a boolean flag variable as true and switch it n times. If flag variable gets original value (which is true) back, then n is even. Else n is false.

Below is the implementation of this idea.

C++

```
 // A simple C++ program to check for
// even or odd
#include <iostream>
using namespace std;

// Returns true if n is even, else odd
bool isEven(int n)
{
    bool isEven = true;
```

```
    for (int i=1;  i <= n; i++)
        isEven = !isEven;
    return isEven;
}

// Driver code
int main()
{
    int n = 101;
    isEven(n) ? cout << "Even" : cout << "Odd";
    return 0;
}
```

Java

```
 // A simple Java program to
// check for even or odd

class GFG {

    // Returns true if n
    // is even, else odd
    static boolean isEven(int n)
    {
        boolean isEven = true;

        for (int i = 1; i <= n; i++)
            isEven = !isEven;

        return isEven;
    }


    // Driver Code
    public static void main(String args[])
    {

        int n = 101;
        if(isEven(n))
            System.out.println("Even");
        else
            System.out.println("Odd");

    }
}

// This code is contributed by Sam007
```

Python3

```
 # A simple Python program to
# check for even or odd

# Returns true if n is even,
# else odd
def isEven(n):
    isEven = True;
    for i in range(1, n+1):
        if isEven == True:
            isEven = False;
        else:
            isEven = True;

    return isEven;

# Driver code
n = 101;
if isEven(n) == True:
    print ("Even");
else:
    print ("Odd");

# This code is contributed by
# Manish Shaw (manishshaw1)
```

C#

```
 // A simple C# program to check for
// even or odd
using System;

public class GFG {

    // Returns true if n is even,
    // else odd
    static bool isEven(int n)
    {
        bool isEven = true;

        for (int i = 1; i <= n; i++)
            isEven = !isEven;

        return isEven;
    }
```

```
    // Driver code
    public static void Main()
    {
        int n = 101;
        if(isEven(n))
            Console.Write("Even");
        else
            Console.Write("Odd");
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// A simple PHP program to check for
// even or odd

// Returns true if n is even,
// else odd
function isEven($n)
{
    $isEven = true;
    for ($i = 1; $i <= $n; $i++)
        $isEven = !$isEven;
    return $isEven;
}

    // Driver code
    $n = 101;
    $is=isEven($n) ? "Even" : "Odd";
    echo "$is"

// This code is contributed by ajit
?>
```

Output:

```
Odd
```

Method 2: By multiply and divide by 2. Divide the number by 2 and multiply by 2 if the result is same as input then it is an even number else it is an odd number.

C++

```
 // A simple C++ program to check for
// even or odd
#include <iostream>
using namespace std;

// Returns true if n is even, else odd
bool isEven(int n)
{
    // Return true if n/2 does not result
    // in a float value.
    return ((n / 2) * 2 == n);
}

// Driver code
int main()
{
    int n = 101;
    isEven(n) ? cout << "Even" : cout << "Odd";
    return 0;
}
```

Java

```
 // A simple Java program
// to check for even or odd

class GFG {

// Returns true if n
// is even, else odd
static boolean isEven(int n)
{

    // Return true if
    // n/2 does not result
    // in a float value.
    return ((n / 2) * 2 == n);
}

// Driver code
public static void main(String[] args)
{
    int n = 101;
    if(isEven(n) != false)
        System.out.print( "Even" );
    else
        System.out.print( "Odd" );
}
```

```
}

// This code is contributed by
// Smitha Dinesh Semwal.
```

Python 3

```
 # A simple Python 3 program
# to check for even or odd

# Returns true if n
# is even, else odd
def isEven(n):


    # Return true if
    # n/2 does not result
    # in a float value.
    return (int(n / 2) * 2 == n)


# Driver code
n = 101
if(isEven(n) != False):
    print("Even")
else:
    print("Odd")

# This code is contributed by
# Smitha Dinesh Semwal.
```

C#

```
 // A simple C# program
// to check for even or odd
using System;
class GFG {

// Returns true if n
// is even, else odd
static bool isEven(int n)
{

    // Return true if
    // n/2 does not result
    // in a float value.
    return ((n / 2) * 2 == n);
```

```
}

// Driver code
public static void Main(String[] args)
{
    int n = 101;
    if(isEven(n) != false)
        Console.Write("Even");
    else
        Console.Write("Odd");
}
}

// This code is contributed by
// Smitha Dinesh Semwal.
```

PHP

```
 <?php
// A simple PHP program to
// check for even or odd

// Returns true if n
// is even, else odd
function isEven($n)
{
    // Return true if n/2
    // does not result
    // in a float value.
    return ((int)($n / 2) * 2 == $n);
}

// Driver code
$n = 101;
if(isEven($n))
    echo ("Even");
else
    echo ("Odd");

// This code is contributed by
// Manish Shaw (manishshaw1)
?>
```

Output :

```
Odd
```

Method 3: Using Bitwise operator &.
A **better solution** is to use bitwise operators. We need to check whether last bit is 1 or not. If last bit is 1 then the number is odd, otherwise always even.

Explanation:

```
 input : 5    // odd
   00000101
 & 00000001
--------------
   00000001
--------------

input : 8     //even
   00001000
 & 00000001
--------------
   00000000
--------------
```

Below is the implementation of the idea.

C++

```
 // A simple C++ program to check for
// even or odd
#include <iostream>
using namespace std;

// Returns true if n is even, else odd
bool isEven(int n)
{
    // n&1 is 1, then odd, else even
    return (!(n & 1));
}

// Driver code
int main()
{
    int n = 101;
    isEven(n) ? cout << "Even" : cout << "Odd";
    return 0;
}
```

Java

```
 // A simple Java program to check for
// even or odd
import java.io.*;
import java.util.*;

public class GFG {

    // Returns 0 if n
    // is even, else odd
    static int isEven(int n)
    {

        // n&1 is 1, then
        // odd, else even
        return (n & 1);
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 101;
        if(isEven(n)==0)
            System.out.print("Even");
        else
            System.out.print("Odd");
    }
}

// This code is contributed
// by Manish Shaw (manishshaw1)
```

Python3

```
 # A simple Python program to
# check for even or odd
# Returns 0 if n
# is even, else odd
def isEven(n) :

    # n&1 is 1, then
    # odd, else even
    return (n & 1);

# Driver code
n = 101;
if(isEven(n) == 0) :
    print ("Even");
else :
```

```
    print ("Odd");

# This code is contributed
# by Manish Shaw (manishshaw1)
```

C#

```
 // A simple C# program to check for
// even or odd
using System;
using System.Collections.Generic;

class GFG {

    // Returns 0 if n
    // is even, else odd
    static int isEven(int n)
    {

        // n&1 is 1, then
        // odd, else even
        return (n & 1);
    }

    // Driver code
    public static void Main()
    {
        int n = 101;
        if(isEven(n)==0)
            Console.Write("Even");
        else
            Console.Write("Odd");
    }
}

// This code is contributed
// by Manish Shaw (manishshaw1)
```

PHP

```
 <?php
// A simple PHP program to
// check for even or odd
// Returns 0 if n
// is even, else odd
function isEven($n)
{
```

```
    // n&1 is 1, then
    // odd, else even
    return ($n & 1);
}

// Driver code
$n = 101;
if(isEven($n) == 0)
    echo ("Even");
else
    echo ("Odd");

// This code is contributed
// by Manish Shaw (manishshaw1)
?>
```

Output :

```
Odd
```

Improved By : Sam007, Smitha Dinesh Semwal, jit_t, manishshaw1

Source

https://www.geeksforgeeks.org/3-ways-check-number-odd-even-without-using-modulus-operator-set-2-can-merge-h

Chapter 38

Check divisibility in a binary stream

Check divisibility in a binary stream - GeeksforGeeks

Stream of binary number is coming, the task is to tell the number formed so far is divisible by a given number n.
At any given time, you will get 0 or 1 and tell whether the number formed with these bits is divisible by n or not.

Generally, e-commerce companies ask this type of questions. It was asked me in Microsoft interview. Actually that question was a bit simple, interviewer fixed the n to 3.

Method 1 (Simple but causes overflow):

Keep on calculating the number formed and just check divisibility by n.

```
 /* Simple implementation of the logic,
   without error handling compiled with
   Microsoft visual studio 2015 */
void CheckDivisibility2(int n)
{
    int num = 0;
    std::cout << "press any key other than"
                " 0 and 1 to terminate \n";
    while (true)
    {
        int incomingBit;

        // read next incoming bit via standard
        // input. 0, 00, 000.. are same as int 0
        // ans 1, 01, 001, 00..001 is same as 1.
```

```
        std::cin >> incomingBit;

        // Update value of num formed so far
        if (incomingBit == 1)
            num = (num * 2 + 1);
        else if (incomingBit == 0)
            num = (num * 2);

        else
            break;
        if (num % n == 0)
            std::cout << "yes \n";
        else
            std::cout << "no \n";
    }
}
```

Problem in this solution: What about the overflow. Since 0 and 1 will keep on coming and the number formed will go out of range of integer.

Method 2 (Doesn't cause overflow) :

In this solution, we just maintain the remainder if remainder is 0, the formed number is divisible by n otherwise not. This is the same technique that is used in Automata to remember the state. Here also we are remembering the state of divisibility of input number.

In order to implement this technique, we need to observe how the value of a binary number changes, when it is appended by 0 or 1.

Let's take an example. Suppose you have binary number 1.
If it is appended by 0 it will become 10 (2 in decimal) means 2 times of the previous value.
If it is appended by 1 it will become 11(3 in decimal), 2 times of previous value +1.

How does it help in getting the remainder?
Any number (n) can be written in the form m = an + r where a, n and r are integers and r is the remainder. So when m is multiplied by any number so the remainder. Suppose m is multiplied by x so m will be mx = xan + xr. so (mx)%n = (xan)%n + (xr)%n = 0 + (xr)%n = (xr)%n;

We need to just do the above calculation (calculation of value of number when it is appended by 0 or 1) only over remainder.

```
When a binary number is appended by 0 (means
multiplied by 2), the new remainder can be
calculated based on current remainder only.
r = 2*r % n;
```

```
And when a binary number is appended by 1.
r = (2*r + 1) % n;


 // C++ program to check divisibility in a stream
#include <iostream>
using namespace std;

/*  A very simple implementation of the logic,
    without error handling. just to demonstrate
    the above theory. This simple version not
    restricting user from typing 000, 00 , 000.. ,
    because this all will be same as 0 for integer
    same is true for 1, 01, 001, 000...001 is same
    as 1, so ignore this type of error handling
    while reading just see the main logic is correct. */
void CheckDivisibility(int n)
{
    int remainder = 0;
    std::cout << "press any key other than 0"
                 " and 1 to terminate \n";
    while (true)
    {
        // Read next incoming bit via standard
        // input. 0, 00, 000.. are same as int 0
        // ans 1, 01, 001, 00..001 is same as 1.
        int incomingBit;
        cin >> incomingBit;

        // Update remainder
        if (incomingBit == 1)
            remainder = (remainder * 2 + 1) % n;
        else if (incomingBit == 0)
            remainder = (remainder * 2) % n;
        else
            break;

        // If remainder is 0.
        if (remainder % n == 0)
            cout << "yes \n";
        else
            cout << "no \n";
    }
}

// Driver code
int main()
{
    CheckDivisibility(3);
```

```
    return 0;
}
```

Input:

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

Output:

```
Press any key other than 0 and 1 to terminate
no
no
no
no
yes
```

Related Articles:
DFA based division
Check if a stream is Multiple of 3

Source

https://www.geeksforgeeks.org/check-divisibility-binary-stream/

Chapter 39

Check divisibility of binary string by 2^k

Check divisibility of binary string by 2^k - GeeksforGeeks

Given a binary string and a number **k**, task is to check whether the binary string is evenly divisible by 2^k or not.

Examples :

```
Input : 11000  k = 2
Output : Yes
Explanation :
(11000)2 = (24)10
24 is evenly divisible by 2k i.e. 4.

Input : 10101  k = 3
Output : No
Explanation :
(10101)2 = (21)10
21 is not evenly divisible by 2k i.e. 8.
```

Naive Approach : Compute the binary string into decimal and then simply check whether it is divisible by 2^k. But, this approach is not feasible for long binary string as time complexity will be high for computing decimal number from binary string and then dividing it by 2^k.

Efficient Approach : In the binary string, check for last k bits. If the all the last k bits are 0, then the binary number is evenly divisible by 2^k else it is not evenly divisible. Time complexity using this approach is O(k).

Below is the implementation of the approach.

C++

```
 // C++ implementation to check whether
// given binary number is evenly
// divisible by 2^k or not
#include <bits/stdc++.h>
using namespace std;

// function to check whether
// given binary number is
// evenly divisible by 2^k or not
bool isDivisible(char str[], int k)
{
    int n = strlen(str);
    int c = 0;

    // count of number of 0 from last
    for (int i = 0; i < k; i++)
        if (str[n - i - 1] == '0')
            c++;

    // if count = k, number is evenly
    // divisible, so returns true else
    // false
    return (c == k);
}

// Driver program to test above
int main()
{
    // first example
    char str1[] = "10101100";
    int k = 2;
    if (isDivisible(str1, k))
        cout << "Yes" << endl;
    else
        cout << "No"
             << "\n";

    // Second example
    char str2[] = "111010100";
    k = 2;
    if (isDivisible(str2, k))
        cout << "Yes" << endl;
    else
        cout << "No" << endl;

    return 0;
}
```

Java

```
 // Java implementation to check whether
// given binary number is evenly
// divisible by 2^k or not
class GFG {

    // function to check whether
    // given binary number is
    // evenly divisible by 2^k or not
    static boolean isDivisible(String str, int k)
    {
        int n = str.length();
        int c = 0;

        // count of number of 0 from last
        for (int i = 0; i < k; i++)
            if (str.charAt(n - i - 1) == '0')
                c++;

        // if count = k, number is evenly
        // divisible, so returns true else
        // false
        return (c == k);
    }

    // Driver program to test above
    public static void main(String args[])
    {

        // first example
        String str1 = "10101100";
        int k = 2;
        if (isDivisible(str1, k) == true)
            System.out.println("Yes");
        else
            System.out.println("No");

    // Second example
        String str2 = "111010100";
        k = 2;
        if (isDivisible(str2, k) == true)
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

```
// This code is contributed by JaideepPyne.
```

Python 3

```
 # Python 3 implementation to check
# whether given binary number is
# evenly divisible by 2^k or not

# function to check whether
# given binary number is
# evenly divisible by 2^k or not
def isDivisible(str, k):
    n = len(str)
    c = 0

    # count of number of 0 from last
    for i in range(0, k):
        if (str[n - i - 1] == '0'):
            c += 1

    # if count = k, number is evenly
    # divisible, so returns true else
    # false
    return (c == k)

# Driver program to test above
# first example
str1 = "10101100"
k = 2
if (isDivisible(str1, k)):
    print("Yes")
else:
    print("No")

# Second example
str2 = "111010100"
k = 2
if (isDivisible(str2, k)):
    print("Yes")
else:
    print("No")

# This code is contributed by Smitha
```

C#

```
 // C# implementation to check whether
```

```
// given binary number is evenly
// divisible by 2^k or not
using System;

class GFG {

    // function to check whether
    // given binary number is
    // evenly divisible by 2^k or not
    static bool isDivisible(String str, int k)
    {
        int n = str.Length;
        int c = 0;

        // count of number of 0 from last
        for (int i = 0; i < k; i++)
            if (str[n - i - 1] == '0')
                c++;

        // if count = k, number is evenly
        // divisible, so returns true else
        // false
        return (c == k);
    }

    // Driver program to test above
    public static void Main()
    {

        // first example
        String str1 = "10101100";
        int k = 2;

        if (isDivisible(str1, k) == true)
            Console.Write("Yes\n");
        else
            Console.Write("No");

        // Second example
        String str2 = "111010100";
        k = 2;

        if (isDivisible(str2, k) == true)
            Console.Write("Yes");
        else
            Console.Write("No");
    }
}
```

```
// This code is contributed by Smitha.
```

PHP

```
 <?php
// PHP implementation to check whether
// given binary number is evenly

// function to check whether
// given binary number is
// evenly divisible by 2^k or not
function isDivisible($str, $k)
{
    $n = strlen($str);
    $c = 0;

    // count of number
    // of 0 from last
    for ($i = 0; $i < $k; $i++)
        if ($str[$n - $i - 1] == '0')
            $c++;

    // if count = k,
    // number is evenly
    // divisible, so
    // returns true else
    // false
    return ($c == $k);
}

    // Driver Code
    // first example
    $str1 = "10101100";
    $k = 2;
    if (isDivisible($str1, $k))
        echo "Yes", "\n";
    else
        echo "No", "\n";

    // Second example
    $str2= "111010100";
    $k = 2;
    if (isDivisible($str2, $k))
        echo "Yes", "\n";
    else
        echo "No", "\n";
```

```
// This code is contributed by Ajit
?>
```

Output:

```
Yes
Yes
```

Time Complexity: O(k)

Improved By : jaideeppyne1997, jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/check-divisibility-binary-string-2k/

Chapter 40

Check for Integer Overflow

Check for Integer Overflow - GeeksforGeeks

Write a “C” function, int addOvf(int* result, int a, int b) If there is no overflow, the function places the resultant = sum a+b in “result” and returns 0. Otherwise it returns -1. The solution of casting to long and adding to find detecting the overflow is not allowed.

Method 1
There can be overflow only if signs of two numbers are same, and sign of sum is opposite to the signs of numbers.

```
1)  Calculate sum
2)  If both numbers are positive and sum is negative then return -1
     Else
        If both numbers are negative and sum is positive then return -1
        Else return 0
```

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

/* Takes pointer to result and two numbers as
    arguments. If there is no overflow, the function
    places the resultant = sum a+b in "result" and
    returns 0, otherwise it returns -1 */
 int addOvf(int* result, int a, int b)
 {
     *result = a + b;
     if(a > 0 && b > 0 && *result < 0)
         return -1;
     if(a < 0 && b < 0 && *result > 0)
         return -1;
     return 0;
```

```
 }

 int main()
 {
     int *res = (int *)malloc(sizeof(int));
     int x = 2147483640;
     int y = 10;

     printf("%d", addOvf(res, x, y));

     printf("\n %d", *res);
     getchar();
     return 0;
}
```

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

Method 2
Thanks to Himanshu Aggarwal for adding this method. This method doesn't modify *result if there us an overflow.

```
 #include<stdio.h>
#include<limits.h>
#include<stdlib.h>

int addOvf(int* result, int a, int b)
{
   if( a > INT_MAX - b)
     return -1;
   else
   {
     *result = a + b;
      return 0;
   }
}

int main()
{
  int *res = (int *)malloc(sizeof(int));
  int x = 2147483640;
  int y = 10;

  printf("%d", addOvf(res, x, y));
  printf("\n %d", *res);
  getchar();
  return 0;
}
```

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

Source

https://www.geeksforgeeks.org/check-for-integer-overflow/

Chapter 41

Check if a given number is sparse or not

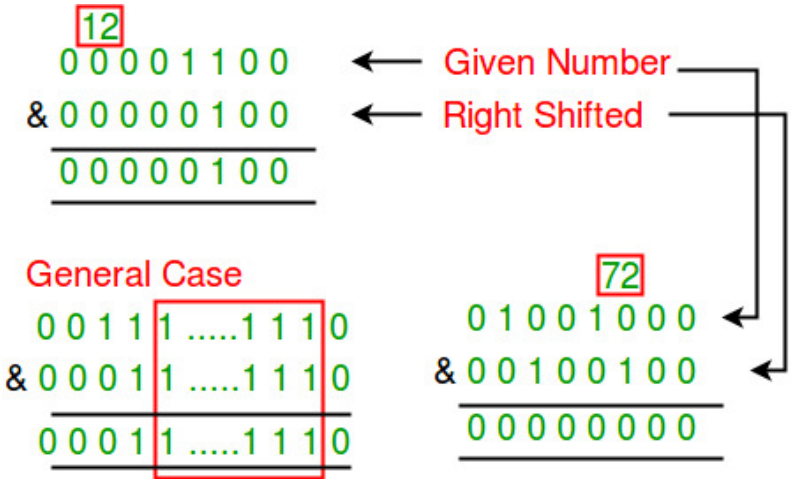
Check if a given number is sparse or not - GeeksforGeeks

A number is said to be a sparse number if in binary representation of the number no two or more consecutive bits are set. Write a function to check if a given number is Sparse or not.

Example :

```
Input:  x  = 72
Output: true
Explanation:  Binary representation of 72 is 01001000.
There are no two consecutive 1's in binary representation

Input:  x  = 12
Output: false
Explanation:  Binary representation of 12 is 1100.
Third and fourth bits (from end) are set.
```

If we observer carefully, then we can notice that if we can use bitwise AND of binary representation of the "given number its "right shifted number"(i.e., half the given number) to figure out whether the number is sparse or not. Result of AND operator would be 0 if number is sparse and non-zero if not sparse.



Below is the implementation of above idea.

C++

```
 // C++ program to check if n is sparse or not
#include<iostream>
using namespace std;

// Return true if n is sparse, else false
bool checkSparse(int n)
{
    // n is not sparse if there is set
    // in AND of n and n/2
    if (n & (n>>1))
        return false;

    return true;
}

// Driver program
int main()
{
    cout << checkSparse(72) << endl;
    cout << checkSparse(12) << endl;
    cout << checkSparse(2) << endl;
    cout << checkSparse(3) << endl;
    return 0;
}
```

Java

```
 // JAVA Code to Check if a
```

```
// given number is sparse or not
import java.util.*;

class GFG {

    // Return true if n is
    // sparse,else false
    static int checkSparse(int n)
    {

        // n is not sparse if there
        // is set in AND of n and n/2
        if ((n & (n>>1)) >=1)
            return 0;

        return 1;
    }

    // Driver code
    public static void main(String[] args)
    {
        System.out.println(checkSparse(72)) ;
        System.out.println(checkSparse(12)) ;
        System.out.println(checkSparse(2)) ;
        System.out.println(checkSparse(3)) ;
        }
    }

//This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python program to check
# if n is sparse or not

# Return true if n is
# sparse, else false
def checkSparse(n):

    # n is not sparse if there is set
    # in AND of n and n/2
    if (n & (n>>1)):
        return 0

    return 1

# Driver code
print(checkSparse(72))
```

```
print(checkSparse(12))
print(checkSparse(2))
print(checkSparse(30))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# Code to Check if a guven
// number is sparse or not
using System;

class GFG {

    // Return true if n is
    // sparse,else false
    static int checkSparse(int n)
    {

        // n is not sparse if there
        // is set in AND of n and n/2
        if ((n & (n >> 1)) >= 1)
            return 0;

        return 1;
    }

    // Driver code
    public static void Main()
    {
        Console.WriteLine(checkSparse(72));
        Console.WriteLine(checkSparse(12));
        Console.WriteLine(checkSparse(2));
        Console.WriteLine(checkSparse(3));
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to check if
// n is sparse or not
// Return true if n is sparse,
// else false
```

```
function checkSparse($n)
{
    // n is not sparse if
    // there is set in AND
    // of n and n/2
    if ($n & ($n >> 1))
        return 0;

    return 1;
}

// Driver Code
echo checkSparse(72), "\n";
echo checkSparse(12), "\n";
echo checkSparse(2), "\n";
echo checkSparse(3), "\n";

// This code is contributed by Ajit.
?>
```

Output :

```
1
0
1
0
```

Note: Instead of right shift, we could have used left shift also, but left shift might lead to overflow in some cases.

This article is contributed by **Vimal Vestron**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : Sam007, jit_t

Source

https://www.geeksforgeeks.org/check-if-a-given-number-is-sparse-or-not/

Chapter 42

Check if a number can be expressed as 2^x + 2^y

Check if a number can be expressed as 2^x + 2^y - GeeksforGeeks

Given a number n, we need to check if it can be expressed as $2^x + 2^y$ or not . Here x and y can be equal.

Examples :

```
Input  : 24
Output : Yes
Explanation: 24 can be expressed as
24 + 23

Input  : 13
output : No
Explanation: It is not possible to
express 13 as sum of two powers of 2.
```

If we take few examples, we can notice that a number can be expressed in the form of 2^x + 2^y if the number is already a power of 2 (for n > 1) or the remainder we get after subtracting previous power of two from the number is also a power of 2.

Below is the implementation of above idea

C++

```
 // CPP code to check if a number can be
// expressed as  2^x + 2^y
#include <bits/stdc++.h>
using namespace std;
```

```
// Utility function to check if
// a number is power of 2 or not
bool isPowerOfTwo(int n)
{
    return (n && !(n & (n - 1)));
}

// Utility function to determine the
// value of previous power of 2
int previousPowerOfTwo(int n)
{
    while (n & n - 1) {
        n = n & n - 1;
    }
    return n;
}

// function to check if n can be expressed
// as 2^x + 2^y or not
bool checkSum(int n)
{
    // if value of n is 0 or 1
    // it can not be expressed as
    // 2^x + 2^y
    if (n == 0 || n == 1)
       return false;

    // if a number is power of 2
    // then it can be expressed as
    // 2^x + 2^y
    else if (isPowerOfTwo(n)) {
        cout << " " << n / 2 << " " << n / 2;
        return true;
    }

    else {
        // if the remainder after
        // subtracting previous power of 2
        // is also a power of 2 then
        // it can be expressed as
        // 2^x + 2^y
        int x = previousPowerOfTwo(n);
        int y = n - x;
        if (isPowerOfTwo(y)) {
            cout << " " << x << " " << y;
            return true;
        }
    }
```

```
    return false;
}

// driver code
int main()
{
    int n1 = 20;
    if (checkSum(n1) == false)
        cout << "No";

    cout << endl;
    int n2 = 11;
    if (checkSum(n2) == false)
        cout << "No";

    return 0;
}
```

Java

```
 // Java code to check if a number
// can be expressed as 2^x + 2^y

class GFG {

    // Utility function to check if
    // a number is power of 2 or not
    static boolean isPowerOfTwo(int n)
    {
        return n != 0 && ((n & (n - 1)) == 0);
    }

    // Utility function to determine the
    // value of previous power of 2
    static int previousPowerOfTwo(int n)
    {
        while ((n & n - 1) > 1) {
            n = n & n - 1;
        }
        return n;
    }

    // function to check if
    // n can be expressed as
    // 2^x + 2^y or not
    static boolean checkSum(int n)
    {
```

```
        // if value of n is 0 or 1
        // it can not be expressed as
        // 2^x + 2^y
        if (n == 0 || n == 1)
            return false;

        // if a number is power of 2
        // it can be expressed as
        // 2^x + 2^y

        else if (isPowerOfTwo(n)) {
            System.out.println(n / 2 + " " + n / 2);
        }
        else {

            // if the remainder after
            // subtracting previous power of 2
            // is also a power of 2 then
            // it can be expressed as
            // 2^x + 2^y
            int x = previousPowerOfTwo(n);
            int y = n - x;
            if (isPowerOfTwo(y)) {

                System.out.println(x + " " + y);
                return true;
            }
        }

         return false;
    }
    // driver code
    public static void main(String[] argc)
    {
    int n1 = 20;
    if (checkSum(n1) == false)
        System.out.println("No");

    System.out.println();
    int n2 = 11;
    if (checkSum(n2) == false)
        System.out.println("No");
    }
}
```

Python3

```
 # Python3 code to check if a number
```

```
# can be expressed as
# 2 ^ x + 2 ^ y

# Utility function to check if
# a number is power of 2 or not
def isPowerOfTwo( n):
    return (n and (not(n & (n - 1))) )


# Utility function to determine the
# value of previous power of 2
def previousPowerOfTwo( n ):
    while( n & n-1 ):
        n = n & n - 1
    return n

# function to check if
# n can be expressed as
# 2 ^ x + 2 ^ y or not
def checkSum(n):

        # if value of n is 0 or 1
        # it can not be expressed as
        # 2 ^ x + 2 ^ y
        if (n == 0 or n == 1 ):
            return False

        # if n is power of two then
        # it can be expressed as
        # sum of 2 ^ x + 2 ^ y
        elif(isPowerOfTwo(n)):
            print(n//2, n//2)
            return True

        # if the remainder after
        # subtracting previous power of 2
        # is also a power of 2 then
        # it can be expressed as
        # 2 ^ x + 2 ^ y
        else:
                x = previousPowerOfTwo(n)
                y = n-x;
                if (isPowerOfTwo(y)):
                    print(x, y)
                    return True
                else:
                    return False
```

```
# driver code
n1 = 20
if (checkSum(n1)):
  print("No")

n2 = 11
if (checkSum(n2)):
  print("No")
```

C#

```
 // C# code to check if a number
// can be expressed as
// 2^x + 2^y

using System;
class GFG {

    // Utility function to check if
    // a number is power of 2 or not
    static bool isPowerOfTwo(int n)
    {
        return n != 0 && ((n & (n - 1)) == 0);
    }

    // Utility function to determine the
    // value of previous power of 2
    static int previousPowerOfTwo(int n)
    {

        while ((n & n - 1) > 1) {
            n = n & n - 1;
        }
        return n;
    }

    // function to check if
    // n can be expressed as
    // 2^x + 2^y or not
    static bool checkSum(int n)
    {
        // if value of n is 0 or 1
        // it can not be expressed as
        // 2^x + 2^y
        if (n == 0 || n == 1) {
            Console.WriteLine("No");
            return false;
        }
```

```
        // if a number is power of
        // it can be expressed as
        // 2^x + 2^y

        else if (isPowerOfTwo(n)) {
            Console.WriteLine(n / 2 + " " + n / 2);
            return true;
        }

        else {

            // if the remainder after
            // subtracting previous power  of 2
            // is also a power of 2 then
            // it can be expressed as
            // 2^x + 2^y

            int x = previousPowerOfTwo(n);
            int y = n - x;
            if (isPowerOfTwo(y)) {
                Console.WriteLine(x + " " + y);
                return true;
            }
            else {
                return false;
            }
        }
    }
    // driver code
    public static void Main()
    {
    int n1 = 20;
    if (checkSum(n1) == false)
        Console.WriteLine("No");

    Console.WriteLine();
    int n2 = 11;
    if (checkSum(n2) == false)
        Console.WriteLine("No");
    }
}
```

PHP

```
 <?php
// PHP code to check if a number
// can be expressed as 2^x + 2^y
```

```
// Utility function to check if
// a number is power of 2 or not
function isPowerOfTwo($n)
{
    return ($n and !($n & ($n - 1)));
}

// Utility function to determine
// the value of previous power of 2
function previousPowerOfTwo($n)
{
    while ($n & $n - 1)
    {
        $n = $n & $n - 1;
    }
    return $n;
}

// function to check if
// n can be expressed
// as 2^x + 2^y or not
function checkSum($n)
{
    // if value of n is 0 or 1
    // it can not be expressed
    // as 2^x + 2^y
    if ($n == 0 or $n == 1)
    return false;

    // if a number is power of 2
    // then it can be expressed
    // as 2^x + 2^y
    else if (isPowerOfTwo($n))
    {
        echo " " , $n / 2 , " " , $n / 2;
        return true;
    }

    else
    {
        // if the remainder after
        // subtracting previous power
        // of 2 is also a power of 2
        // then it can be expressed
        // as 2^x + 2^y
        $x = previousPowerOfTwo($n);
        $y = $n - $x;
```

```
        if (isPowerOfTwo($y))
        {
            echo $x, " ", $y;
            return true;
        }
    }

    return false;
}

// Driver code
$n1 = 20;
if (checkSum($n1) == false)
    echo "No";
echo"\n";

$n2 = 11;
if (checkSum($n2) == false)
    echo "No";

// This code is contributed
// by chandan_jnu.
?>
```

Output:

```
16 4
No
```

Improved By : Chandan_Kumar

Source

https://www.geeksforgeeks.org/check-if-a-number-can-be-expressed-as-2x-2y/

Chapter 43

Check if a number can be expressed as a sum of consecutive numbers

Check if a number can be expressed as a sum of consecutive numbers - GeeksforGeeks

Given a number n, the task is to check whether it can be expressed as a sum of two or more consecutive numbers or not.

Examples:

```
Input  : n = 10
Output : true
It can be expressed as sum of two consecutive
numbers 1 + 2 + 3 + 4.

Input  : n = 16
Output : false
It cannot be expressed as sum of two consecutive
numbers.

Input  : n = 5
Output : true
2 + 3 = 5
```

There is a direct and quick method to solve this. If a number is a power of two, then it cannot be expressed as a sum of consecutive numbers otherwise Yes.

The idea is based on below two facts.
1) Sum of any two consecutive numbers is odd as one of them has to be even and other odd.
2) $2^n = 2^{n-1} + 2^{n-1}$

If we take a closer look at 1) and 2), we can get intuition behind the fact.

Below is implementation of above idea.

C++

```
 // C++ program to check if a number can
// be expressed as sum of consecutive numbers
#include<bits/stdc++.h>
using namespace std;

// This function returns true if n can be
// expressed sum of consecutive.
bool canBeSumofConsec(unsigned int n)
{
    // We basically return true if n is a
    // power of two
    return ((n&(n-1)) && n);
}

// Driver code
int main()
{
    unsigned int n = 15;
    canBeSumofConsec(n)? cout << "true" :
                         cout << "false";
    return 0;
}
```

Java

```
 // Java program to check if a number can
// be expressed as sum of consecutive numbers

class Test
{
    // This function returns true if n can be
    // expressed sum of consecutive.
    static boolean canBeSumofConsec(int n)
    {
        // We basically return true if n is a
        // power of two
        return (((n&(n-1))!=0) && n!=0);
    }

    // Driver method
    public static void main(String[] args)
```

```
    {
        int n = 15;
        System.out.println(canBeSumofConsec(n) ? "true" : "false");
    }
}
```

Python3

```
 # Python 3 program to check if a number can
# be expressed as sum of consecutive numbers


# This function returns true if n
# can be expressed sum of consecutive.
def canBeSumofConsec(n) :

    # We basically return true if n is a
    # power of two
    return ((n&(n-1)) and n)


# Driver code
n = 15
if(canBeSumofConsec(n)) :
    print("true")
else :
    print("false")

# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# program to check if a number can be
// expressed as sum of consecutive numbers
using System;

class Test
{
    // This function returns true if n
    // can be expressed sum of consecutive.
    static bool canBeSumofConsec(int n)
    {
        // We basically return true if n is a
        // power of two
        return (((n & (n - 1)) != 0) && n != 0);
    }
```

```
    // Driver Code
    public static void Main()
    {
        int n = 15;
        Console.Write(canBeSumofConsec(n) ? "True" : "False");
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// php program to check if a number
// can be expressed as sum of
// consecutive numbers

// This function returns true if n
// can be expressed sum of consecutive.
function canBeSumofConsec($n)
{

    // We basically return true if n is a
    // power of two
    return (($n & ($n - 1)) && $n);
}

// Driver code
    $n = 15;
    if(canBeSumofConsec($n))
        echo "true" ;
    else
        echo "false";

// This code is contributed by
// nitin mittal.
?>
```

Output:

```
True
```

Reference:
http://www.cut-the-knot.org/arithmetic/UnpropertyOfPowersOf2.shtml

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/check-number-can-expressed-sum-consecutive-numbers/

Chapter 44

Check if a number has bits in alternate pattern | Set 1

Check if a number has bits in alternate pattern | Set 1 - GeeksforGeeks

Given an integer n > 0, the task is to find whether this integer has an alternate pattern in its bits representation. For example- 5 has an alternate pattern i.e. 101.
Print "Yes" if it has an alternate pattern otherwise "No". Here alternate pattern can be like 0101 or 1010.

Examples:

```
Input :  15
Output :  No
Explanation: Binary representation of 15 is 1111.

Input :  10
Output :  Yes
Explanation: Binary representation of 10 is 1010.
```

A simple approach is to find its binary equivalent and then check its bits.

C++

```
 // C++ program to find if a number has alternate
// bit pattern
#include<bits/stdc++.h>
using namespace std;

// Returns true if n has alternate bit pattern
// else false
```

```
bool findPattern(int n)
{
    // Store last bit
    int prev = n % 2;
    n = n/2;

    // Traverse through remaining bits
    while (n > 0)
    {
        int curr = n % 2;

        // If current bit is same as previous
        if (curr == prev)
           return false;

        prev = curr;
        n = n / 2;
    }

    return true;
}

// Driver code
int main()
{
    int n = 10;
    if (findPattern(n))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
 // Java program to find if a number has alternate
// bit pattern

class Test
{
    // Returns true if n has alternate bit pattern
    // else false
    static boolean findPattern(int n)
    {
        // Store last bit
        int prev = n % 2;
        n = n/2;
```

```
        // Traverse through remaining bits
        while (n > 0)
        {
            int curr = n % 2;

            // If current bit is same as previous
            if (curr == prev)
               return false;

            prev = curr;
            n = n / 2;
        }

        return true;
    }

    // Driver method
    public static void main(String args[])
    {
        int n = 10;
        System.out.println(findPattern(n) ? "Yes" : "No");

    }
}
```

Python3

```
 # Python3 program to find if a number
# has alternate bit pattern

# Returns true if n has alternate
# bit pattern else false
def findPattern(n):

    # Store last bit
    prev = n % 2
    n = n // 2

    # Traverse through remaining bits
    while (n > 0):

        curr = n % 2

        # If current bit is same as previous
        if (curr == prev):
            return False
```

```
        prev = curr
        n = n // 2

    return True

# Driver code
n = 10
print("Yes") if(findPattern(n)) else print("No")

# This code is contributed by Anant Agarwal.
```

C#

```
 // Program to find if a number
// has alternate bit pattern
using System;

class Test {

    // Returns true if n has alternate
    // bit pattern else returns false
    static bool findPattern(int n)
    {
        // Store last bit
        int prev = n % 2;
        n = n / 2;

        // Traverse through remaining bits
        while (n > 0) {
            int curr = n % 2;

            // If current bit is same as previous
            if (curr == prev)
                return false;

            prev = curr;
            n = n / 2;
        }

        return true;
    }

    // Driver method
    public static void Main()
    {
        int n = 10;
        Console.WriteLine(findPattern(n) ? "Yes" : "No");
    }
```

```
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program to find if a
// number has alternate
// bit pattern

// Returns true if n has
// alternate bit pattern
// else false
function findPattern($n)
{

    // Store last bit
    $prev = $n % 2;
    $n = $n / 2;

    // Traverse through
    // remaining bits
    while ($n > 0)
    {
        $curr = $n % 2;

        // If current bit is
        // same as previous
        if ($curr == $prev)
        return false;

        $prev = $curr;
        $n = floor($n / 2);
    }

    return true;
}

    // Driver code
    $n = 10;
    if (findPattern($n))
        echo "Yes";
    else
        echo "No";

    return 0;
```

```
// This code is contributed by nitin mittal.
?>
```

Output:

```
Yes
```

Reference:
http://stackoverflow.com/questions/38690278/program-to-check-whether-the-given-integer-has-an-alternate-patte

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/check-if-a-number-has-bits-in-alternate-pattern/

Chapter 45

Check if a number has bits in alternate pattern | Set-2 O(1) Approach

Check if a number has bits in alternate pattern | Set-2 O(1) Approach - GeeksforGeeks

Given a positive integer **n**. The problem is to check whether this integer has an alternate pattern in its binary representation or not. Here alternate pattern means that the set and unset bits in **n** occur in alternate order. For example- 5 has an alternate pattern i.e. 101. Print "Yes" if it has an alternate pattern otherwise "No".

Note: 0 < n.

Examples :

```
Input : 10
Output : Yes
(10)10 = (1010)2, has an alternate pattern.

Input : 12
Output : No
(12)10 = (1100)2, does not have an alternate pattern.
```

Simple Approach: It has been discussed in this post having a time complexity of O(n).

Efficient Approach: Following are the steps:

1. Calculate **num** = n ^ (n >> 1). If **n** has an alternate pattern, then **n ^ (n >> 1)** operation will produce a number having set bits only. '^' is a bitwise XOR operation.
2. Check whether all the bits in **num** are set or not. Refer this post.

C++

```
 // C++ implementation to check if a number
// has bits in alternate pattern
#include <bits/stdc++.h>

using namespace std;

// function to check if all the bits are set or not
// in the binary representation of 'n'
bool allBitsAreSet(unsigned int n)
{
    // if true, then all bits are set
    if (((n + 1) & n) == 0)
        return true;

    // else all bits are not set
    return false;
}

// function to check if a number
// has bits in alternate pattern
bool bitsAreInAltOrder(unsigned int n)
{
    unsigned int num = n ^ (n >> 1);

    // to check if all bits are set
    // in 'num'
    return allBitsAreSet(num);
}

// Driver program to test above
int main()
{
    unsigned int n = 10;

    if (bitsAreInAltOrder(n))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
 // Java implementation to check if a
```

```
// number has bits in alternate pattern
class AlternateSetBits
{
    // function to check if all the bits
    // are set or not in the binary
    // representation of 'n'
    static boolean allBitsAreSet(int n)
    {
        // if true, then all bits are set
        if (((n + 1) & n) == 0)
            return true;

        // else all bits are not set
        return false;
    }

    // function to check if a number
    // has bits in alternate pattern
    static boolean bitsAreInAltOrder(int n)
    {
        int num = n ^ (n >>> 1);

        // to check if all bits are set
        // in 'num'
        return allBitsAreSet(num);
    }

    // Driver Code
    public static void main(String args[])
    {
        int n = 10;

        if (bitsAreInAltOrder(n))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
/* This code is contributed by Danish Kaleem */
```

C#

```
 // C# implementation to check if a
// number has bits in alternate pattern
using System;

class GFG {
```

```
    // function to check if all the bits
    // are set or not in the binary
    // representation of 'n'
    static bool allBitsAreSet(int n)
    {
        // if true, then all bits are set
        if (((n + 1) & n) == 0)
            return true;

        // else all bits are not set
        return false;
    }

    // function to check if a number
    // has bits in alternate pattern
    static bool bitsAreInAltOrder(int n)
    {
        int num = n ^ (n >> 1);

        // to check if all bits are set
        // in 'num'
        return allBitsAreSet(num);
    }

    // Driver Code
    public static void Main()
    {
        int n = 10;

        if (bitsAreInAltOrder(n))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }

}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP implementation to check
// if a number has bits in
// alternate pattern

// function to check if all the
// bits are set or not in the
```

```
// binary representation of 'n'
function allBitsAreSet($n)
{
    // if true, then all
    // bits are set
    if ((($n + 1) & $n) == 0)
        return true;

    // else all bits are not set
    return false;
}

// function to check if a number
// has bits in alternate pattern
function bitsAreInAltOrder($n)
{
        $num = $n ^ ($n >> 1);

    // to check if all bits
    // are set in 'num'
    return allBitsAreSet($num);
}

// Driver Code
$n = 10;

if (bitsAreInAltOrder($n))
    echo "Yes";
else
    echo "No";

// This code is contributed by aj_36
?>
```

Output :

```
Yes
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/check-number-bits-alternate-pattern-set-2-o1-approach/

Chapter 46

Check if a number has same number of set and unset bits

Check if a number has same number of set and unset bits - GeeksforGeeks

Given a number N, the task is to check whether the count of the set and unset bits in the given number are same.

Examples:

```
Input: 12
Output: Yes
1100 is the binary representation of 12
which has 2 set and 2 unset bits

Input: 14
Output: No
```

Approach: Traverse in the binary representation of the given number, check if the leftmost bit is set or not using **n & 1**. If n & 1 returns 1, then the left most bit is set. Right, shift the number every time by 1 to check the next bit. Once the binary representation is traversed completely, check if the number of set bit and unset bits are same. If they are same, print "YES" else print "NO".

Below is the implementation of the above approach:

C++

```
 // C++ program to check if a number
// has same number of set and unset bits
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to check if a number
// has same setbits and unset bits
bool checkSame(int n)
{
    int set = 0, unset = 0;

    // iterate for all bits of a number
    while (n) {

        // if set
        if (n & 1)
            set++;
        // if unset
        else
            unset++;

        // right shift number by 1
        n = n >> 1;
    }

    // is number of set bits are
    // equal to unset bits
    if (set == unset)
        return true;
    else
        return false;
}

// Driver Code
int main()
{
    int n = 12;

    // function to check
    if (checkSame(n))
        cout << "YES";
    else
        cout << "NO";
    return 0;
}
```

Java

```
 // Java program to check if
// a number has same number
// of set and unset bits
import java.io.*;
```

```
class GFG
{

// Function to check if a
// number has same setbits
// and unset bits
static boolean checkSame(int n)
{
    int set = 0;
    int unset = 0;

    // iterate for all
    // bits of a number
    while (n == 0)
    {

        // if set
        if (n > 1)
            set++;

        // if unset
        else
            unset++;

        // right shift
        // number by 1
        n = n >> 1;
    }

    // is number of set bits
    // are equal to unset bits
    if (set == unset)
        return true;
    else
        return false;
}

// Driver Code
public static void main (String[] args)
{
int n = 12;

// function to check
if (checkSame(n))
    System.out.println ("YES");
else
    System.out.println("NO");
```

```
}
}

// This code is contributed
// by akt_mit
```

Output:

```
YES
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/check-if-a-number-has-same-number-of-set-and-unset-bits/

Chapter 47

Check if a number has two adjacent set bits

Check if a number has two adjacent set bits - GeeksforGeeks

Given a number you have to check whether there is pair of adjacent set bit or not.

Examples :

```
Input : N = 67
Output : Yes
There is a pair of adjacent set bit
The binary representation is 100011

Input : N = 5
Output : No
```

A **simple solution** is traverse all bits. For every set bit, check if next bit is also set.

An efficient solution is to shift number by 1 and then do bitwise AND. If bitwise AND is non-zero then there are two adjacent set bits. Else not.

C++

```
 // CPP program to check
// if there are two
// adjacent set bits.
#include <iostream>
using namespace std;

bool adjacentSet(int n)
{
```

```
    return (n & (n >> 1));
}

// Driver Code
int main()
{
    int n = 3;
    adjacentSet(n) ?
     cout << "Yes" :
       cout << "No";
    return 0;
}
```

Java

```
 // Java program to check
// if there are two
// adjacent set bits.
class GFG
{

    static boolean adjacentSet(int n)
    {
        int x = (n & (n >> 1));

        if(x == 1)
            return true;
        else
            return false;
    }

    // Driver code
    public static void main(String args[])
    {

        int n = 3;

        if(adjacentSet(n))
            System.out.println("Yes");
        else
            System.out.println("No");

    }
}

// This code is contributed by Sam007.
```

C#

```
 // C# program to check
// if there are two
// adjacent set bits.
using System;

class GFG
{
    static bool adjacentSet(int n)
    {
        int x = (n & (n >> 1));

        if(x == 1)
            return true;
        else
            return false;
    }

    // Driver code
    public static void Main ()
    {
        int n = 3;

        if(adjacentSet(n))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }

}

// This code is contributed by Sam007.
```

php

```
 <?php
// PHP program to check
// if there are two
// adjacent set bits.

function adjacentSet($n)
{
    return ($n & ($n >> 1));
}

// Driver Code
$n = 3;
adjacentSet($n) ?
   print("Yes") :
```

```
    print("No");

// This code is contributed by Sam007.
?>
```

Output :

```
Yes
```

Improved By : Sam007

Source

https://www.geeksforgeeks.org/check-number-two-adjacent-set-bits/

Chapter 48

Check if a number is Bleak

Check if a number is Bleak - GeeksforGeeks

A number 'n' is called Bleak if it cannot be represented as sum of a positive number x and set bit count in x, i.e., x + countSetBits(x) is not equal to n for any non-negative number x.

Examples :

```
Input : n = 3
Output : false
3 is not Bleak as it can be represented
as 2 + countSetBits(2).

Input : n = 4
Output : true
4 is t Bleak as it cannot be represented
as sum of a number x and countSetBits(x)
for any number x.
```

Method 1 (Simple)

```
bool isBleak(n)
1) Consider all numbers smaller than n
    a) If x + countSetBits(x) == n
           return false

2) Return true
```

Below is the implementation of the simple approach.

C++

```
 // A simple C++ program to check Bleak Number
#include <bits/stdc++.h>
using namespace std;

/* Function to get no of set bits in binary
   representation of passed binary no. */
int countSetBits(int x)
{
    unsigned int count = 0;
    while (x) {
        x &= (x - 1);
        count++;
    }
    return count;
}

// Returns true if n is Bleak
bool isBleak(int n)
{
    // Check for all numbers 'x' smaller
    // than n.  If x + countSetBits(x)
    // becomes n, then n can't be Bleak
    for (int x = 1; x < n; x++)
        if (x + countSetBits(x) == n)
            return false;

    return true;
}

// Driver code
int main()
{
    isBleak(3) ? cout << "Yes\n" : cout << "No\n";
    isBleak(4) ? cout << "Yes\n" : cout << "No\n";
    return 0;
}
```

Java

```
 // A simple Java program to check Bleak Number
import java.io.*;

class GFG {

    /* Function to get no of set bits in binary
       representation of passed binary no. */
    static int countSetBits(int x)
    {
```

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

    // Returns true if n is Bleak
    static boolean isBleak(int n)
    {
        // Check for all numbers 'x' smaller
        // than n.  If x + countSetBits(x)
        // becomes n, then n can't be Bleak
        for (int x = 1; x < n; x++)
            if (x + countSetBits(x) == n)
                return false;

        return true;
    }

    // Driver code
    public static void main(String args[])
    {
        if (isBleak(3))
            System.out.println("Yes");
        else
            System.out.println("No");
        if (isBleak(4))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

/*This code is contributed by Nikita Tiwari.*/
```

Python3

```
 # A simple Python 3 program
# to check Bleak Number

# Function to get no of set
# bits in binary
# representation of passed
# binary no.
def countSetBits(x) :
```

```
    count = 0

    while (x) :
        x = x & (x-1)
        count = count + 1

    return count

# Returns true if n
# is Bleak
def isBleak(n) :

    # Check for all numbers 'x'
    # smaller than n. If x +
    # countSetBits(x) becomes
    # n, then n can't be Bleak.
    for x in range(1, n) :

        if (x + countSetBits(x) == n) :
            return False

    return True

# Driver code
if(isBleak(3)) :
    print( "Yes")
else :
    print("No")

if(isBleak(4)) :
    print("Yes")
else :
    print( "No")

# This code is contributed by Nikita Tiwari.
```

C#

```
 // A simple C# program to check
// Bleak Number
using System;

class GFG {

    /* Function to get no of set
    bits in binary representation
    of passed binary no. */
    static int countSetBits(int x)
```

```
    {
        int count = 0;

        while (x != 0) {
            x &= (x - 1);
            count++;
        }

        return count;
    }

    // Returns true if n is Bleak
    static bool isBleak(int n)
    {

        // Check for all numbers
        // 'x' smaller than n. If
        // x + countSetBits(x)
        // becomes n, then n can't
        // be Bleak
        for (int x = 1; x < n; x++)

            if (x + countSetBits(x)
                                == n)
                return false;

        return true;
    }

    // Driver code
    public static void Main()
    {
        if (isBleak(3))
            Console.Write("Yes");
        else
            Console.WriteLine("No");

        if (isBleak(4))
            Console.Write("Yes");
        else
            Console.Write("No");
    }
}

// This code is contributed by
// Nitin mittal
```

PHP

```
 <?php
// A simple PHP program
// to check Bleak Number

// Function to get no of
// set bits in binary
// representation of
// passed binary no.
function countSetBits( $x)
{
    $count = 0;
    while ($x)
    {
        $x &= ($x - 1);
        $count++;
    }
    return $count;
}

// Returns true if n is Bleak
function isBleak( $n)
{

    // Check for all numbers 'x' smaller
    // than n. If x + countSetBits(x)
    // becomes n, then n can't be Bleak
    for($x = 1; $x < $n; $x++)
        if ($x + countSetBits($x) == $n)
            return false;

    return true;
}

    // Driver code
    if(isBleak(3))
        echo "Yes\n" ;
    else
        echo "No\n";

    if(isBleak(4))
        echo "Yes\n" ;
    else
        echo "No\n";

// This code is contributed by anuj_67.
?>
```

Output :

```
No
Yes
```

Time complexity of above solution is O(n Log n).

Method 2 (Efficient)
The idea is based on the fact that the largest count of set bits in any number smaller than n cannot exceed ceiling of $Log_2 n$. So we need to check only numbers from range n – ceilingLog2(n) to n.

```
bool isBleak(n)
1) Consider all numbers n - ceiling(Log2n) to n-1
    a) If x + countSetBits(x) == n
           return false

2) Return true
```

Below is the implementation of the idea.

C++

```
 // An efficient C++ program to check Bleak Number
#include <bits/stdc++.h>
using namespace std;

/* Function to get no of set bits in binary
   representation of passed binary no. */
int countSetBits(int x)
{
    unsigned int count = 0;
    while (x) {
        x &= (x - 1);
        count++;
    }
    return count;
}

// A function to return ceiling of log x
// in base 2. For example, it returns 3
// for 8 and 4 for 9.
int ceilLog2(int x)
{
    int count = 0;
    x--;
```

```
    while (x > 0) {
        x = x >> 1;
        count++;
    }
    return count;
}

// Returns true if n is Bleak
bool isBleak(int n)
{
    // Check for all numbers 'x' smaller
    // than n.  If x + countSetBits(x)
    // becomes n, then n can't be Bleak
    for (int x = n - ceilLog2(n); x < n; x++)
        if (x + countSetBits(x) == n)
            return false;

    return true;
}

// Driver code
int main()
{
    isBleak(3) ? cout << "Yes\n" : cout << "No\n";
    isBleak(4) ? cout << "Yes\n" : cout << "No\n";
    return 0;
}
```

Java

```
 // An efficient Java program to
// check Bleak Number
import java.io.*;

class GFG {

    /* Function to get no of set bits in
    binary representation of passed binary
    no. */
    static int countSetBits(int x)
    {
        int count = 0;
        while (x != 0) {
            x &= (x - 1);
            count++;
        }
        return count;
    }
```

```
    // A function to return ceiling of log x
    // in base 2. For example, it returns 3
    // for 8 and 4 for 9.
    static int ceilLog2(int x)
    {
        int count = 0;
        x--;
        while (x > 0) {
            x = x >> 1;
            count++;
        }
        return count;
    }

    // Returns true if n is Bleak
    static boolean isBleak(int n)
    {
        // Check for all numbers 'x' smaller
        // than n. If x + countSetBits(x)
        // becomes n, then n can't be Bleak
        for (int x = n - ceilLog2(n); x < n; x++)
            if (x + countSetBits(x) == n)
                return false;

        return true;
    }

    // Driver code
    public static void main(String[] args)
    {
        if (isBleak(3))
            System.out.println("Yes");
        else
            System.out.println("No");

        if (isBleak(4))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
// This code is contributed by Prerna Saini
```

Python3

```
 # An efficient Python 3 program
# to check Bleak Number
```

```
import math

# Function to get no of set
# bits in binary representation
# of passed binary no.
def countSetBits(x) :

    count = 0

    while (x) :
        x = x & (x - 1)
        count = count + 1

    return count

# A function to return ceiling
# of log x in base 2. For
# example, it returns 3 for 8
# and 4 for 9.
def ceilLog2(x) :

    count = 0
    x = x - 1

    while (x > 0) :
        x = x>>1
        count = count + 1

    return count

# Returns true if n is Bleak
def isBleak(n) :

    # Check for all numbers 'x'
    # smaller than n. If x +
    # countSetBits(x) becomes n,
    # then n can't be Bleak
    for x in range ((n - ceilLog2(n)), n) :

        if (x + countSetBits(x) == n) :
            return False

    return True

# Driver code
if(isBleak(3)) :
    print("Yes")
else :
```

```
    print( "No")

if(isBleak(4)) :
    print("Yes")
else :
    print("No")

# This code is contributed by Nikita Tiwari.
```

C#

```
 // An efficient C# program to check
// Bleak Number
using System;

class GFG {

    /* Function to get no of set
    bits in binary representation
    of passed binary no. */
    static int countSetBits(int x)
    {
        int count = 0;
        while (x != 0) {
            x &= (x - 1);
            count++;
        }
        return count;
    }

    // A function to return ceiling
    // of log x in base 2. For
    // example, it returns 3 for 8
    // and 4 for 9.
    static int ceilLog2(int x)
    {
        int count = 0;
        x--;
        while (x > 0) {
            x = x >> 1;
            count++;
        }
        return count;
    }

    // Returns true if n is Bleak
    static bool isBleak(int n)
    {
```

```
        // Check for all numbers
        // 'x' smaller than n. If
        // x + countSetBits(x)
        // becomes n, then n
        // can't be Bleak
        for (int x = n - ceilLog2(n);
                          x < n; x++)
            if (x + countSetBits(x)
                               == n)
                return false;

        return true;
    }

    // Driver code
    public static void Main()
    {
        if (isBleak(3))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");

        if (isBleak(4))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// An efficient PHP program
// to check Bleak Number

/* Function to get no of set
   bits in binary representation
   of passed binary no. */
function countSetBits( $x)
{
    $count = 0;
    while ($x)
    {
        $x &= ($x - 1);
        $count++;
```

```
    }
    return $count;
}

// A function to return ceiling of log x
// in base 2. For example, it returns 3
// for 8 and 4 for 9.
function ceilLog2( $x)
{

    $count = 0;
    $x--;
    while ($x > 0)
    {
        $x = $x >> 1;
        $count++;
    }
    return $count;
}

// Returns true if n is Bleak
function isBleak( $n)
{

    // Check for all numbers 'x' smaller
    // than n. If x + countSetBits(x)
    // becomes n, then n can't be Bleak
    for ($x = $n - ceilLog2($n); $x < $n; $x++)
        if ($x + countSetBits($x) == $n)
            return false;

    return true;
}

    // Driver code
    if(isBleak(3))
        echo "Yes\n" ;

    else
        echo "No\n";

    if(isBleak(4))
        echo "Yes\n" ;

    else
        echo "No\n";

// This code is contributed by anuj_67
```

```
?>
```

Output :

```
No
Yes
```

Time Complexity: O(Log n * Log n)

Note: In GCC, we can directly count set bits using ___builtin_popcount(). So we can avoid a separate function for counting set bits.

```
 // C++ program to demonstrate __builtin_popcount()
#include <iostream>
using namespace std;

int main()
{
    cout << __builtin_popcount(4) << endl;
    cout << __builtin_popcount(15);

    return 0;
}
```

Output :

```
1
4
```

This article is contributed by **Rahuain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/check-if-a-number-is-bleak/

Chapter 49

Check if a number is divisible by 17 using bitwise operators

Check if a number is divisible by 17 using bitwise operators - GeeksforGeeks

Given a number n, check if it is divisible by 17 using bitwise operators.

Examples:

```
Input : n = 34
Output : 34 is divisible by 17

Input :  n = 43
Output : 43 is not divisible by 17
```

A **naive approach** will be to check it by % operator if it leaves a remainder of 0.

To do division using **Bitwise operators**, we must rewrite the expression in powers of 2.

```
n/17 = (16*n)/(17*16)
     = (17 - 1)*n/(17*16)
     = (n/16) - (n/(17*16))
```

We can rewrite n/16 as floor(n/16) + (n%16)/16 using general rule of division.

```
n/17 = floor(n/16) + (n%16)/16 -
       (floor(n/16) + (n%16)/16)/17
     = floor(n/16) - (floor(n/16) -
            17*(n%16)/16 + (n%16)/16)/17
     = floor(n/16) - (floor(n/16)-n%16)/17
```

The left-hand-side of this equation is n/17. That will be an integer only when the right-hand-side is an integer. floor(n/16) is an integer by definition. So the whole left-hand-side would be an integer if (floor(n/16)-n%16)/17 is also an integer.

This implies n is divisible by 17 if (floor(n/16)-n%16) is divisible by 17.

(floor(n/16)-n%16) can be written in bitwise as **(int)(n>>4)** – **(int)(n&15)** where n>>4 means n/16 and n%15 means n%15

Below is the implementation of the above approach:

CPP

```
 // CPP program to check if a number is
// divisible by 17 or not using bitwise
// operator.
#include <bits/stdc++.h>
using namespace std;

// function to check recursively if the
// number is divisible by 17 or not
bool isDivisibleby17(int n)
{
    // if n=0 or n=17 then yes
    if (n == 0 || n == 17)
        return true;

    // if n is less then 17, not
    // divisible by 17
    if (n < 17)
        return false;

    // reducing the number by floor(n/16)
    // - n%16
    return isDivisibleby17((int)(n >> 4) - (int)(n & 15));
}

// driver code to check the above function
int main()
{
    int n = 35;
    if (isDivisibleby17(n))
        cout << n << " is divisible by 17";
    else
        cout << n << " is not divisible by 17";
    return 0;
}
```

Java

```
 // Java program to check if a number is
// divisible by 17 or not using bitwise
// operator.
class GFG{

    // function to check recursively if the
    // number is divisible by 17 or not
    static boolean isDivisibleby17(int n)
    {

        // if n=0 or n=17 then yes
        if (n == 0 || n == 17)
            return true;

        // if n is less then 17, not
        // divisible by 17
        if (n < 17)
            return false;

        // reducing the number by
        // floor(n/16) - n%16
        return isDivisibleby17((int)(n >> 4)
                            - (int)(n & 15));
    }

    // driver function
    public static void main(String[] args)
    {
        int n = 35;
        if (isDivisibleby17(n) == true)
            System.out.printf
            ("%d is divisible by 17",n);
        else
            System.out.printf
            ("%d is not divisible by 17",n);
    }
}

// This code is contributed by
// Smitha Dinesh Semwal
```

Python3

```
 # Python 3 program to
# check if a number is
# divisible by 17 or
# not using bitwise
# operator.
```

```
# function to check recursively if the
# number is divisible by 17 or not
def isDivisibleby17(n):

    # if n=0 or n=17 then yes
    if (n == 0 or n == 17):
        return True

    # if n is less then 17, not
    # divisible by 17
    if (n < 17):
        return False

    # reducing the number by floor(n/16)
    # - n%16
    return isDivisibleby17((int)(n >> 4) - (int)(n & 15))


# driver code to check the above function
n = 35
if (isDivisibleby17(n)):
    print(n,"is divisible by 17")
else:
    print(n,"is not divisible by 17")

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to check if a number is
// divisible by 17 or not using bitwise
// operator.
using System;

class GFG
{

    // function to check recursively if the
    // number is divisible by 17 or not
    static bool isDivisibleby17(int n)
    {

        // if n=0 or n=17 then yes
        if (n == 0 || n == 17)
            return true;
```

```
        // if n is less then 17, not
        // divisible by 17
        if (n < 17)
            return false;

        // reducing the number by
        // floor(n/16) - n%16
        return isDivisibleby17((int)(n >> 4)
                            - (int)(n & 15));
    }

    // Driver function
    public static void Main()
    {
        int n = 35;
        if (isDivisibleby17(n) == true)
            Console.WriteLine
            (n +"is divisible by 17");
        else
            Console.WriteLine
            ( n+ " is not divisible by 17");
    }
}

// This code is contributed by
// vt_m
```

PHP

```
 <?php
// php program to check if a
// number is divisible by 17
// or not using bitwise
// operator.

// function to check recursively
// if the number is divisible
// by 17 or not
function isDivisibleby17($n)
{

    // if n=0 or n=17 then yes
    if ($n == 0 || $n == 17)
        return true;

    // if n is less then 17, not
    // divisible by 17
    if ($n < 17)
```

```
        return false;

    // reducing the number by floor(n/16)
    // - n%16
    return isDivisibleby17((int)($n >> 4) -
                           (int)($n & 15));
}

    // Driver Code
    $n = 35;
    if (isDivisibleby17($n))
        echo $n." is divisible by 17";
    else
        echo $n." is not divisible by 17";

// This code is contributed by mits
?>
```

Output:

```
35 is not divisible by 17
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/check-number-divisible-17-using-bitwise-operators/

Chapter 50

Check if a number is divisible by 8 using bitwise operators

Check if a number is divisible by 8 using bitwise operators - GeeksforGeeks

Given a number n, check if it is divisible by 8 using bitwise operators.

Examples:

```
Input : 16
Output :YES

Input :15
Output :NO
```

Approach: Result = (((n >> 3) << 3) == n). First we shift the 3 bit right then we shift the 3 bit left and then compare the number with the given number if the number is equal to the number then it is the divisible by 8 .
Explanation:
Example: n = 16 given so binary of the 16 is 10000 now we shift the 3 bit right, now we have 00010 so again we shift the 3 bit left, then we have 10000 now compare with the given number first 16==16 in binary so it true so the number is divisible by 8.

CPP

```
 // C++ program to check whether
// the number is divisible by
// 8 or not using bitwise operator
#include <bits/stdc++.h>
using namespace std;

// function to check number is
```

```
// div by 8 or not using bitwise
// operator
int Div_by_8(int n)
{
    return (((n >> 3) << 3) == n);
}

// Driver program
int main()
{
    int n = 16;
    if (Div_by_8(n))
        cout << "YES" << endl;
    else
        cout << "NO" << endl;
    return 0;
}
```

Java

```
 // Java program to check whether
// the number is divisible by
// 8 or not using bitwise operator
import java.io.*;
import java.util.*;

class GFG
{
    // function to check number is
    // div by 8 or not using bitwise
    // operator
    static boolean Div_by_8(int n)
    {
        return (((n >> 3) << 3) == n);
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 16;
        if (Div_by_8(n))
            System.out.println("YES");
        else
            System.out.println("NO");

    }
}
```

```
// This code is contributed by  Gitanjali
```

Python3

```
 # Python  program to check whether
# the number is divisible by
# 8 or not using bitwise operator

import math

# function to check number is
# div by 8 or not using bitwise
# operator

def Div_by_8(n):

    return (((n >> 3) << 3) == n)


#  driver code
n = 16
if (Div_by_8(n)):
    print("YES")
else:
    print("NO")

# This code is contributed by Gitanjali.
```

C#

```
// C# program to check whether
// the number is divisible by
// 8 or not using bitwise operator
using System;
class GFG {

 // function to check number is
 // div by 8 or not using bitwise
 // operator
 static bool Div_by_8(int n)
 {
 return (((n >> 3) << 3) == n);
 }

 // Driver code
 public static void Main ()
```

```
 {
 int n = 16;

 if (Div_by_8(n))
 Console.WriteLine("YES");
 else
 Console.WriteLine("NO");

 }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to check whether
// the number is divisible by
// 8 or not using bitwise operator

// function to check number is
// div by 8 or not using bitwise
// operator
function Div_by_8($n)
{
    return ((($n >> 3) << 3) == $n);
}

// Driver program
$n = 16;

if (Div_by_8($n))
    echo "YES";
else
    echo "NO";

//This code is contributed by mits.
?>
```

```
YES
```

Improved By : AnirudhTiwari, Mithun Kumar

Source

https://www.geeksforgeeks.org/check-number-divisible-8-using-bitwise-operators/

Chapter 51

Check if a number is multiple of 9 using bitwise operators

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

Given a number n, write a function that returns true if n is divisible by 9, else false. The most simple way to check for n's divisibility by 9 is to do n%9.
Another method is to sum the digits of n. If sum of digits is multiple of 9, then n is multiple of 9.
The above methods are not bitwise operators based methods and require use of % and /.
The bitwise operators are generally faster than modulo and division operators. Following is a bitwise operator based method to check divisibility by 9.

C++

```
 // C++ program to check if a number
// is multiple of 9 using bitwise operators
#include <bits/stdc++.h>
using namespace std;

// Bitwise operator based function to check divisibility by 9
bool isDivBy9(int n)
{
    // Base cases
    if (n == 0 || n == 9)
        return true;
    if (n < 9)
        return false;

    // If n is greater than 9, then recur for [floor(n/9) - n%8]
    return isDivBy9((int)(n >> 3) - (int)(n & 7));
}
```

```
// Driver program to test above function
int main()
{
    // Let us print all multiples of 9 from 0 to 100
    // using above method
    for (int i = 0; i < 100; i++)
        if (isDivBy9(i))
            cout << i << " ";
    return 0;
}
```

Java

```
 // Java program to check if a number
// is multiple of 9 using bitwise operators
import java.lang.*;

class GFG {

    // Bitwise operator based function
    // to check divisibility by 9
    static boolean isDivBy9(int n)
    {

        // Base cases
        if (n == 0 || n == 9)
            return true;
        if (n < 9)
            return false;

        // If n is greater than 9, then
        // recur for [floor(n/9) - n%8]
        return isDivBy9((int)(n >> 3) - (int)(n & 7));
    }

    // Driver code
    public static void main(String arg[])
    {

        // Let us print all multiples of 9 from
        // 0 to 100 using above method
        for (int i = 0; i < 100; i++)
            if (isDivBy9(i))
                System.out.print(i + " ");
    }
}
```

```
// This code is contributed by Anant Agarwal.
```

Python3

```
 # Bitwise operator based
# function to check divisibility by 9

def isDivBy9(n):

    # Base cases
    if (n == 0 or n == 9):
        return True
    if (n < 9):
        return False

    # If n is greater than 9,
    # then recur for [floor(n / 9) - n % 8]
    return isDivBy9((int)(n>>3) - (int)(n&7))

# Driver code

# Let us print all multiples
# of 9 from 0 to 100
# using above method
for i in range(100):
    if (isDivBy9(i)):
        print(i, " ", end ="")

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to check if a number
// is multiple of 9 using bitwise operators
using System;

class GFG {

    // Bitwise operator based function
    // to check divisibility by 9
    static bool isDivBy9(int n)
    {
        // Base cases
        if (n == 0 || n == 9)
            return true;
        if (n < 9)
```

```
            return false;

        // If n is greater than 9, then
        // recur for [floor(n/9) - n%8]
        return isDivBy9((int)(n >> 3) - (int)(n & 7));
    }

    // Driver code
    public static void Main()
    {
        // Let us print all multiples of 9 from
        // 0 to 100 using above method
        for (int i = 0; i < 100; i++)
            if (isDivBy9(i))
                Console.Write(i + " ");
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to check if a number
// is multiple of 9 using bitwise
// operators

// Bitwise operator based function
// to check divisibility by 9
function isDivBy9($n)
{

    // Base cases
    if ($n == 0 || $n == 9)
        return true;
    if ($n < 9)
        return false;

    // If n is greater than 9,
    // then recur for [floor(n/9) -
    // n%8]
    return isDivBy9(($n >> 3) -
                    ($n & 7));
}

    // Driver Code
    // Let us print all multiples
    // of 9 from 0 to 100
```

```
    // using above method
    for ($i = 0; $i < 100; $i++)
        if (isDivBy9($i))
            echo $i ," ";

// This code is contributed by nitin mittal
?>
```

Output:

```
0 9 18 27 36 45 54 63 72 81 90 99
```

How does this work?
n/9 can be written in terms of *n/8* using the following simple formula.

```
n/9 = n/8 - n/72
```

Since we need to use bitwise operators, we get the value of *floor(n/8)* using *n>>3* and get value of *n%8* using *n&7*. We need to write above expression in terms of *floor(n/8)* and *n%8*. *n/8* is equal to *"floor(n/8) + (n%8)/8"*. Let us write the above expression in terms of *floor(n/8)* and *n%8*

```
n/9 = floor(n/8) + (n%8)/8 - [floor(n/8) + (n%8)/8]/9
n/9 = floor(n/8) - [floor(n/8) - 9(n%8)/8 + (n%8)/8]/9
n/9 = floor(n/8) - [floor(n/8) - n%8]/9
```

From above equation, *n* is a multiple of *9* only if the expression *floor(n/8) – [floor(n/8) – n%8]/9* is an integer. This expression can only be an integer if the sub-expression *[floor(n/8) – n%8]/9* is an integer. The subexpression can only be an integer if *[floor(n/8) – n%8]* is a multiple of *9*. So the problem reduces to a smaller value which can be written in terms of bitwise operators.

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/divisibility-9-using-bitwise-operators/

Chapter 52

Check if a number is positive, negative or zero using bit operators

Check if a number is positive, negative or zero using bit operators - GeeksforGeeks

Given a number N, check if it is positive, negative or zero without using conditional statements.

Examples:

```
Input : 30
Output : 30 is positive

Input : -20
Output : -20 is negative

Input: 0
Output: 0 is zero
```

The signed shift n>>31 converts every negative number into -1 and every other into 0.
When we do a -n>>31, if it is a positive number then it will return -1 as we are doing -n>>31 and the vice versa when we do for a negative number.
But when we do for 0 then n>>31 and -n>>31 both returns 0, so we get a formula:

1 + (n>>31) – (-n>>31)

..1) **when n is negative** :
1 +(-1)-0=0

..2) **when n is positive:**
1+0-(-1)=2

..3) **when n is 0:**
1+0-0=1

So we know it returns 0 when it is a negative number, it returns 1 when it is zero, returns 2 when it is a positive number.

So to not use if statements we store at 0th index “negative”, 1st index “zero” and at 2nd index “positive”, and print according to it.

CPP

```
 // CPP program to find if a number is
// positive, negative or zero using
// bit wise operators.
#include <iostream>
using namespace std;

// function to return 1 if it is zero
// returns 0 if it is negative
// returns 2 if it is positive
int index(int i)
{
    return 1 + (i >> 31) - (-i >> 31);
}

void check(int n)
{
    // string array to store all kinds of number
    string s[] = { "negative", "zero", "positive" };

    // function call to check the sign of number
    int val = index(n);

    cout << n << " is " << s[val] << endl;
}

// driver program to test the above function
int main()
{
    check(30);
    check(-20);
    check(0);
    return 0;
}
```

Java

```
 // Java program to find if a number is
// positive, negative or zero using
// bit wise operators.
class GFG {

    // function to return 1 if it is zero
    // returns 0 if it is negative
    // returns 2 if it is positive
    static int index(int i)
    {
        return 1 + (i >> 31) - (-i >> 31);
    }

    static void check(int n)
    {

        // string array to store all kinds
        // of number
        String s[] = { "negative", "zero",
                               "positive" };

        // function call to check the sign
        // of number
        int val = index(n);

        System.out.println(n + " is " + s[val]);
    }

    // Driver code
    public static void main(String[] args)
    {
        check(30);
        check(-20);
        check(0);
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python 3 program to
# find if a number is
# positive, negative
# or zero using
# bit wise operators.

# function to return 1 if it is zero
```

```
# returns 0 if it is negative
# returns 2 if it is positive
def index(i):

    return 1 + (i >> 31) - (-i >> 31)


def check(n):

    # string array to store all kinds of number
    s = "negative", "zero", "positive"

    # function call to check the sign of number
    val = index(n)

    print(n,"is",s[val])


# driver program to
# test the above function
check(30)
check(-20)
check(0)

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to find if a number is
// positive, negative or zero using
// bit wise operators.
using System;

class GFG {

    // function to return 1 if it is zero
    // returns 0 if it is negative
    // returns 2 if it is positive
    static int index(int i)
    {
        return 1 + (i >> 31) - (-i >> 31);
    }

    static void check(int n)
    {

        // string array to store all kinds of number
```

```
        String []s = { "negative", "zero", "positive" };

        // function call to check the sign of number
        int val = index(n);

        Console.WriteLine(n + " is " + s[val]);
    }

    //Driver code
    public static void Main()
    {
        check(30);
        check(-20);
        check(0);
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program to find if a number is
// positive, negative or zero using
// bit wise operators.

// function to return 1 if it is zero
// returns 0 if it is negative
// returns 2 if it is positive
function index($i)
{
    return 1 + ($i >> 31) -
               (-$i >> 31);
}

function check($n)
{

    // string array to store
    // all kinds of number
    $s = array("negative", "zero", "positive" );

    // function call to check
    // the sign of number
    $val = index($n);

    echo $n, " is ", $s[$val], "\n";
}
```

```
    // Driver Code
    check(30);
    check(-20);
    check(0);

// This code is contributed by Ajit
?>
```

Output:

```
30 is positive
-20 is negative
0 is zero
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/check-number-positive-negative-zero-using-bit-operators/

Chapter 53

Check if a number is power of 8 or not

Check if a number is power of 8 or not - GeeksforGeeks

Given a number check whether it is power of 8 or not.

Examples :

```
Input : n = 64
Output : Yes

Input : 75
Output : No
```

First solution

We calculate log8(n) of the number if it is an integer, then n is in power of 8. We use trunc(n) function that finds closest integer for a double value.

C++

```
 // CPP program to check if a number is power of 8
#include <iostream>
#include <cmath>
using namespace std;

/* function to check if power of 8 */
bool checkPowerof8(int n)
{
    /* calculate log8(n) */
    double i = log(n) / log(8);
```

```
    /* check if i is an integer or not */
    return (i - trunc(i) < 0.000001)
}

/* driver function */
int main()
{
    int n = 65;
    checkPowerof8(n) ? cout << "Yes" : cout << "No";
    return 0;
}
```

Java

```
 // Java program to check if
// a number is power of 8

class GFG
{

    // function to check
    // if power of 8
    static boolean checkPowerof8(int n)
    {
        /* calculate log8(n) */
        double i = Math.log(n) / Math.log(8);

        /* check if i is an integer or not */
        return (i - Math.floor(i) < 0.000001);
    }

    // Driver Code
    public static void main(String args[])
    {
        int n = 65;
        if(checkPowerof8(n))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

// This code is contributed by Sam007
```

Output :

```
No
```

Second solution

A number is power of 8 if following conditions are satisfied.

1. The number is power of two. A number is power of two if it has only one set bit, i.e., bitwise and of n and n-1 is 0.
2. The number has its only set bit at position 0 or 3 or 6 or …. 30 [For a 32 bit number]. To check the position of its set bit we can use a mask $(0xB6DB6DB6)_{16} = (10110110110110110110110110110110)_2$.

Below is implementation of above idea.

C++

```
 // C++ program to check if a number is power of 8
// using bit mask.
#include <bits/stdc++.h>
using namespace std;

/*function to check if power of 8*/
bool checkPowerof8(int n)
{
    return (n && !(n & (n - 1)) && !(n & 0xB6DB6DB6));
}

/*driver function*/
int main()
{
    int n = 65;
    checkPowerof8(n) ? cout << "Yes" : cout << "No";

    return 0;
}
```

Output :

```
No
```

Improved By : Sam007

Source

https://www.geeksforgeeks.org/check-number-power-8-not/

Chapter 54

Check if actual binary representation of a number is palindrome

Check if actual binary representation of a number is palindrome - GeeksforGeeks

Given a non-negative integer **n**. The problem is to check if binary representation of **n** is palindrome or not. Note that the actual binary representation of the number is being considered for palindrome checking, no leading 0's are being considered.

Examples :

```
Input : 9
Output : Yes
(9)10 = (1001)2

Input : 10
Output : No
(10)10 = (1010)2
```

Approach: Following are the steps:

1. Get the number obtained by reversing the bits in the binary representation of **n**. Refer this post. Let it be **rev**.
2. If n == rev, then print "Yes" else "No".

C++

```
// C++ implementation to check whether binary
```

```
// representation of a number is palindrome or not
#include <bits/stdc++.h>
using namespace std;

// function to reverse bits of a number
unsigned int reverseBits(unsigned int n)
{
    unsigned int rev = 0;

    // traversing bits of 'n' from the right
    while (n > 0) {

        // bitwise left shift 'rev' by 1
        rev <<= 1;

        // if current bit is '1'
        if (n & 1 == 1)
            rev ^= 1;

        // bitwise right shift 'n' by 1
        n >>= 1;
    }

    // required number
    return rev;
}

// function to check whether binary representation
// of a number is palindrome or not
bool isPalindrome(unsigned int n)
{
    // get the number by reversing bits in the
    // binary representation of 'n'
    unsigned int rev = reverseBits(n);

    return (n == rev);
}

// Driver program to test above
int main()
{
    unsigned int n = 9;
    if (isPalindrome(n))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // Java code  to check whether
// binary representation of a
// number is palindrome or not
import java.util.*;
import java.lang.*;

public class GfG
{
    // function to reverse bits of a number
    public static long reverseBits(long n)
    {
        long rev = 0;

        // traversing bits of 'n'
        // from the right
        while (n > 0)
        {
            // bitwise left shift 'rev' by 1
            rev <<= 1;

            // if current bit is '1'
            if ((n & 1) == 1)
                rev ^= 1;

            // bitwise right shift 'n' by 1
            n >>= 1;
        }

        // required number
        return rev;
    }

    // function to check a number
    // is palindrome or not
    public static boolean isPalindrome(long n)
    {
        // get the number by reversing
        // bits in the  binary
        // representation of 'n'
        long rev = reverseBits(n);

        return (n == rev);
    }

    // Driver function
    public static void main(String argc[])
```

```
    {
        long n = 9;
        if (isPalindrome(n))
            System.out.println("Yes");
        else
            System.out.println("No");
    }

}

// This code is contributed by Sagar Shukla
```

Python3

```
 # Python 3 implementation to check
# whether binary representation of
# a number is palindrome or not

# function to reverse bits of a number
def reverseBits(n) :
    rev = 0

  # traversing bits of 'n' from the right
  while (n > 0) :

     # bitwise left shift 'rev' by 1
     rev = rev << 1

     # if current bit is '1'
     if (n & 1 == 1) :
     rev = rev ^ 1

     # bitwise right shift 'n' by 1
     n = n >> 1

     # required number
     return rev

# function to check whether binary
# representation of a number is
# palindrome or not
def isPalindrome(n) :

    # get the number by reversing
    # bits in the binary
    # representation of 'n'
    rev = reverseBits(n)
```

```
    return (n == rev)


# Driver program to test above
n = 9
if (isPalindrome(n)) :
    print("Yes")
else :
    print("No")

# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# code to check whether
// binary representation of a
// number is palindrome or not
using System;

public class GfG
{
    // function to reverse bits of a number
    public static long reverseBits(long n)
    {
        long rev = 0;

        // traversing bits of 'n'
        // from the right
        while (n > 0)
        {
            // bitwise left shift 'rev' by 1
            rev <<= 1;

            // if current bit is '1'
            if ((n & 1) == 1)
                rev ^= 1;

            // bitwise right shift 'n' by 1
            n >>= 1;
        }

        // required number
        return rev;
    }

    // function to check a number
    // is palindrome or not
    public static bool isPalindrome(long n)
```

```
    {
        // get the number by reversing
        // bits in the binary
        // representation of 'n'
        long rev = reverseBits(n);

        return (n == rev);
    }

    // Driver function
    public static void Main()
    {
        long n = 9;
        if (isPalindrome(n))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }

}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP implementation to check
// whether binary representation
// of a number is palindrome or not

// function to reverse bits of a number
function reverseBits($n)
{
    $rev = 0;

    // traversing bits of 'n'
    // from the right
    while ($n > 0)
    {

        // bitwise left shift 'rev' by 1
        $rev <<= 1;

        // if current bit is '1'
        if ($n & 1 == 1)
            $rev ^= 1;

        // bitwise right shift 'n' by 1
```

```
        $n >>= 1;
    }

    // required number
    return $rev;
}

// function to check whether
// binary representation of a
// number is palindrome or not
function isPalindrome($n)
{
    // get the number by reversing
    // bits in the binary representation
    // of 'n'
    $rev = reverseBits($n);

    return ($n == $rev);
}

// Driver code
$n = 9;
if (isPalindrome($n))
    echo "Yes";
else
    echo "No";
return 0;

// This code is contributed by mits
?>
```

Output :

```
Yes
```

Time Complexity: O(num), where **num** is the number of bits in the binary representation of **n**.

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/check-actual-binary-representation-number-palindrome/

Chapter 55

Check if all bits of a number are set

Check if all bits of a number are set - GeeksforGeeks

Given a number **n**. The problem is to check whether every bit in the binary representation of the given number is set or not. Here 0 <= **n**.

Examples :

```
Input : 7
Output : Yes
(7)10 = (111)2

Input : 14
Output : No
```

Method 1: If **n** = 0, then answer is 'No'. Else perform the two operations until **n** becomes 0.

```
While (n > 0)
 If n & 1 == 0,
     return 'No'
 n >> 1
```

If loop terminates without returning 'No', then all bits are set in the binary representation of **n**.

C++

```
 // C++ implementation to check whether every
// digit in the binary representation of the
// given number is set or not
#include <bits/stdc++.h>
using namespace std;

// function to check if all the bits are set
// or not in the binary representation of 'n'
string areAllBitsSet(int n)
{
    // all bits are not set
    if (n == 0)
        return "No";

    // loop till n becomes '0'
    while (n > 0)
    {
        // if the last bit is not set
        if ((n & 1) == 0)
            return "No";

        // right shift 'n' by 1
        n = n >> 1;
    }

    // all bits are set
    return "Yes";
}

// Driver program to test above
int main()
{
    int n = 7;
    cout << areAllBitsSet(n);
    return 0;
}
```

Java

```
 // java implementation to check
// whether every digit in the
// binary representation of the
// given number is set or not
import java.io.*;

class GFG {

    // function to check if all the bits
```

```
    // are setthe bits are set or not
    // in the binary representation of 'n'
    static String areAllBitsSet(int n)
    {
        // all bits are not set
        if (n == 0)
            return "No";

        // loop till n becomes '0'
        while (n > 0)
        {
            // if the last bit is not set
            if ((n & 1) == 0)
                return "No";

            // right shift 'n' by 1
            n = n >> 1;
        }

            // all bits are set
            return "Yes";
    }

    // Driver program to test above
    public static void main (String[] args) {
    int n = 7;

    System.out.println(areAllBitsSet(n));
    }
}


// This code is contributed by vt_m
```

Python3

```
 # Python implementation
# to check whether every
# digit in the binary
# representation of the
# given number is set or not

# function to check if
# all the bits are set
# or not in the binary
# representation of 'n'
def areAllBitsSet(n):
```

```
    # all bits are not set
    if (n == 0):
        return "No"

    # loop till n becomes '0'
    while (n > 0):

        # if the last bit is not set
        if ((n & 1) == 0):
            return "No"

        # right shift 'n' by 1
        n = n >> 1


    # all bits are set
    return "Yes"


# Driver program to test above

n = 7
print(areAllBitsSet(n))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# implementation to check
// whether every digit in the
// binary representation of the
// given number is set or not
using System;

class GFG
{

    // function to check if
    // all the bits are set
    // or not in the binary
    // representation of 'n'
    static String areAllBitsSet(int n)
    {
        // all bits are not set
        if (n == 0)
            return "No";
```

```
        // loop till n becomes '0'
        while (n > 0)
        {
            // if the last bit
            // is not set
            if ((n & 1) == 0)
                return "No";

            // right shift 'n' by 1
            n = n >> 1;
        }

            // all bits are set
            return "Yes";
    }

    // Driver Code
    static public void Main ()
    {
        int n = 7;
        Console.WriteLine(areAllBitsSet(n));
    }
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP implementation to check
// whether every digit in the
// binary representation of the
// given number is set or not

// function to check if all the
// bits are set or not in the
// binary representation of 'n'
function areAllBitsSet($n)
{
    // all bits are not set
    if ($n == 0)
        return "No";

    // loop till n becomes '0'
    while ($n > 0)
    {
        // if the last bit is not set
        if (($n & 1) == 0)
```

```
            return "No";

        // right shift 'n' by 1
        $n = $n >> 1;
    }

    // all bits are set
    return "Yes";
}

// Driver Code
$n = 7;
echo areAllBitsSet($n);

// This code is contributed by aj_36
?>
```

Output :

```
Yes
```

Time Complexity : O(d), where 'd' is the number of bits in the binary representation of **n**.

Method 2: If **n** = 0, then answer is 'No'. Else add 1 to **n**. Let it be **num** = n + 1. If num & (num – 1) == 0, then all bits are set, else all bits are not set.
Explanation: If all bits in the binary representation of **n** are set, then adding '1' to it will produce a number which will be a perfect power of 2. Now, check whether the new number is a perfect power of 2 or not.

C++

```
 // C++ implementation to check whether every
// digit in the binary representation of the
// given number is set or not
#include <bits/stdc++.h>
using namespace std;

// function to check if all the bits are set
// or not in the binary representation of 'n'
string areAllBitsSet(int n)
{
    // all bits are not set
    if (n == 0)
        return "No";
```

```
    // if true, then all bits are set
    if (((n + 1) & n) == 0)
        return "Yes";

    // else all bits are not set
    return "No";
}

// Driver program to test above
int main()
{
    int n = 7;
    cout << areAllBitsSet(n);
    return 0;
}
```

Java

```
 // JAVA implementation to check whether
// every digit in the binary representation
// of the given number is set or not
import java.io.*;

class GFG {

    // function to check if all the
    // bits are set or not in the
    // binary representation of 'n'
    static String areAllBitsSet(int n)
    {
        // all bits are not set
        if (n == 0)
            return "No";

        // if true, then all bits are set
        if (((n + 1) & n) == 0)
            return "Yes";

        // else all bits are not set
        return "No";
    }

    // Driver program to test above
    public static void main (String[] args) {
    int n = 7;
    System.out.println(areAllBitsSet(n));
    }
```

```
}

// This code is contributed by vt_m
```

Python3

```
 # Python implementation to
# check whether every
# digit in the binary
# representation of the
# given number is set or not

# function to check if
# all the bits are set
# or not in the binary
# representation of 'n'
def areAllBitsSet(n):

    # all bits are not set
    if (n == 0):
        return "No"

    # if true, then all bits are set
    if (((n + 1) & n) == 0):
        return "Yes"

    # else all bits are not set
    return "No"


# Driver program to test above

n = 7
print(areAllBitsSet(n))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# implementation to check
// whether every digit in the
// binary representation of
// the given number is set or not
using System;

class GFG
```

```
{

    // function to check if all the
    // bits are set or not in the
    // binary representation of 'n'
    static String areAllBitsSet(int n)
    {
        // all bits are not set
        if (n == 0)
            return "No";

        // if true, then all
        // bits are set
        if (((n + 1) & n) == 0)
            return "Yes";

        // else all bits are not set
        return "No";
    }

    // Driver Code
    static public void Main ()
    {
        int n = 7;
        Console.WriteLine(areAllBitsSet(n));
    }
}

// This code is contributed by m_kit
```

PHP

```
 <?php
// PHP implementation to check
// whether every digit in the
// binary representation of the
// given number is set or not

// function to check if all
// the bits are set or not in
// the binary representation of 'n'
function areAllBitsSet($n)
{
    // all bits are not set
    if ($n == 0)
        return "No";

    // if true, then all
```

```
    // bits are set
    if ((($n + 1) & $n) == 0)
        return "Yes";

    // else all bits
    // are not set
    return "No";
}

// Driver Code
$n = 7;
echo areAllBitsSet($n);

// This code is contributed by ajit
?>
```

Output :

```
Yes
```

References:
https://www.careercup.com/question?id=9503107

Improved By : jit_t

Source

https://www.geeksforgeeks.org/check-bits-number-set/

Chapter 56

Check if binary representation of a given number and its complement are anagram

Check if binary representation of a given number and its complement are anagram - GeeksforGeeks

Given a positive number you need to check whether it's complement and the number are anagrams or not.

Examples:

```
Input : a = 4294967295
Output : Yes
Binary representation of 'a' and it's
complement are anagrams of each other

Input : a = 4
Output : No
```

Simple Approach: In this approach calculation of the complement of the number is allowed.

1. Find binary representation of the number and it's complement using simple decimal to binary representation technique.

2. Sort both the binary representations and compare them to check whether they are anagrams or not.

```
 // A simple C++ program to check if binary
// representations of a number and it's
```

```
// complement are anagram.
#include <bits/stdc++.h>
#define ull unsigned long long int
using namespace std;

const int ULL_SIZE = 8*sizeof(ull);

bool isComplementAnagram(ull a)
{
    ull b = ~a; // Finding complement of a;

    // Find reverse binary representation of a.
    bool binary_a[ULL_SIZE] = { 0 };
    for (int i=0; a > 0; i++)
    {
        binary_a[i] = a % 2;
        a /= 2;
    }

    // Find reverse binary representation
    // of complement.
    bool binary_b[ULL_SIZE] = { 0 };
    for (int i=0; b > 0; i++)
    {
        binary_b[i] = b % 2;
        b /= 2;
    }

    // Sort binary representations and compare
    // after sorting.
    sort(binary_a, binary_a + ULL_SIZE);
    sort(binary_b, binary_b + ULL_SIZE);
    for (int i = 0; i < ULL_SIZE; i++)
        if (binary_a[i] != binary_b[i])
            return false;

    return true;
}

// Driver code
int main()
{
    ull a = 4294967295;
    cout << isComplementAnagram(a) << endl;
    return 0;
}
```

Output:

```
1
```

Efficient Approach: Just count the number of 1's present in the bit representation of the given number. If number of 1's present are 32 then it's complement will also have 32 1's in it's bit representation and they will be anagrams of each other.

```
 // An efficient C++ program to check if binary
// representations of a number and it's complement are anagram.
#include <bits/stdc++.h>
#define ull unsigned long long int
using namespace std;

const int ULL_SIZE = 8*sizeof(ull);

// Returns true if binary representations of
// a and b are anagram.
bool bit_anagram_check(ull a)
{
    // _popcnt64(a) gives number of 1's present
    // in binary representation of a. If number
    // of 1s is half of total bits, return true.
    return (_popcnt64(a) == (ULL_SIZE >> 1));
}

int main()
{
    ull a = 4294967295;
    cout << bit_anagram_check(a) << endl;
    return 0;
}
```

Output:

```
1
```

Note:
1. The answer is only dependent on the number, in the above approach we don't even find the need to obtain the complement of the number.

2. The above code uses GCC specific functions. If we wish to write code for other compilers, we may use Count set bits in an integer.

Source

https://www.geeksforgeeks.org/check-binary-representation-given-number-complement-anagram/

Chapter 57

Check if binary representation of a number is palindrome

Check if binary representation of a number is palindrome - GeeksforGeeks

Given an integer 'x', write a C function that returns true if binary representation of x is palindrome else return false.

For example a numbers with binary representation as 10..01 is palindrome and number with binary representation as 10..00 is not palindrome.

The idea is similar to checking a string is palindrome or not. We start from leftmost and rightmost bits and compare bits one by one. If we find a mismatch, then return false.

Algorithm:
isPalindrome(x)
1) Find number of bits in x using sizeof() operator.
2) Initialize left and right positions as 1 and n respectively.
3) Do following while left 'l' is smaller than right 'r'.
.....a) If bit at position 'l' is not same as bit at position 'r', then return false.
.....b) Increment 'l' and decrement 'r', i.e., do l++ and r—.
4) If we reach here, it means we didn't find a mismatching bit.

To find the bit at a given position, we can use the idea similar to thispost. The expression "**x & (1 << (k-1))**" gives us non-zero value if bit at k'th position from right is set and gives a zero value if if k'th bit is not set.

Following is C++ implementation of the above algorithm.

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

// This function returns true if k'th bit in x is set (or 1).
// For example if x (0010) is 2 and k is 2, then it returns true
bool isKthBitSet(unsigned int x, unsigned int k)
```

```
{
    return (x & (1 << (k-1)))? true: false;
}

// This function returns true if binary representation of x is
// palindrome. For example (1000...001) is paldindrome
bool isPalindrome(unsigned int x)
{
    int l = 1; // Initialize left position
    int r = sizeof(unsigned int)*8; // initialize right position

    // One by one compare bits
    while (l < r)
    {
        if (isKthBitSet(x, l) != isKthBitSet(x, r))
            return false;
        l++;     r--;
    }
    return true;
}

// Driver program to test above function
int main()
{
    unsigned int x = 1<<15 + 1<<16;
    cout << isPalindrome(x) << endl;
    x = 1<<31 + 1;
    cout << isPalindrome(x) << endl;
    return 0;
}
```

Output:

```
1
1
```

This article is contributed by **Saurabh Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/check-binary-representation-number-palindrome/

Chapter 58

Check if binary representations of two numbers are anagram

Check if binary representations of two numbers are anagram - GeeksforGeeks

Given two numbers you are required to check whether they are anagrams of each other or not in binary representation.

Examples:

```
Input : a = 8, b = 4
Output : Yes
Binary representations of both
numbers have same 0s and 1s.

Input : a = 4, b = 5
Output : No
```

Simple Approach:

1. Find Binary Representation of 'a' and 'b' using simple decimal to binary representation technique.
2. Check if two binary representations are anagram

C/C++

```
 // A simple C++ program to check if binary
// representations of two numbers are anagram.
#include <bits/stdc++.h>
#define ull unsigned long long int
using namespace std;
```

```
const int SIZE = 8 * sizeof(ull);

bool bit_anagram_check(ull a, ull b)
{
    // Find reverse binary representation of a
    // and store it in binary_a[]
    int i = 0, binary_a[SIZE] = { 0 };
    while (a > 0) {
        binary_a[i] = a % 2;
        a /= 2;
        i++;
    }

    // Find reverse binary representation of b
    // and store it in binary_a[]
    int j = 0, binary_b[SIZE] = { 0 };
    while (b > 0) {
        binary_b[j] = b % 2;
        b /= 2;
        j++;
    }

    // Sort two binary representations
    sort(binary_a, binary_a + SIZE);
    sort(binary_b, binary_b + SIZE);

    // Compare two sorted binary representations
    for (int i = 0; i < SIZE; i++)
        if (binary_a[i] != binary_b[i])
            return false;

    return true;
}

// Driver code
int main()
{
    ull a = 8, b = 4;
    cout << bit_anagram_check(a, b) << endl;
    return 0;
}
```

Java

```
 // A simple Java program to check if binary
// representations of two numbers are anagram
import java.io.*;
```

```
import java.util.*;

class GFG
{
    public static int SIZE = 8;

    // Function to check if binary representation
    // of two numbers are anagram
    static int bit_anagram_check(long a, long b)
    {
        // Find reverse binary representation of a
        // and store it in binary_a[]
        int i = 0;
        long[] binary_a = new long[SIZE];
        Arrays.fill(binary_a, 0);
        while (a > 0)
        {
            binary_a[i] = a%2;
            a /= 2;
            i++;
        }

        // Find reverse binary representation of b
        // and store it in binary_a[]
        int j = 0;
        long[] binary_b = new long[SIZE];
        Arrays.fill(binary_b, 0);
        while (b > 0)
        {
            binary_b[j] = b%2;
            b /= 2;
            j++;
        }

        // Sort two binary representations
        Arrays.sort(binary_a);
        Arrays.sort(binary_b);

        // Compare two sorted binary representations
        for (i = 0; i < SIZE; i++)
            if (binary_a[i] != binary_b[i])
                return 0;

        return 1;
    }

    // driver program
    public static void main (String[] args)
```

```
    {
        long a = 8, b = 4;
        System.out.println(bit_anagram_check(a, b));
    }
}

// Contributed by Pramod Kumar
```

Output:

```
1
```

Time Complexity : O (n log n)
Auxiliary Space : O (1) Although Auxiliary Space is O(1) still SIZE array spaces are getting used to store binary representation of each number.

Efficient Approach:
Just measure the number of 1's present in the bit representation of both the numbers, if number of 1's present in their bit representation are same then they are anagrams in their bit representation else they are not.

C/C++

```
 // An efficient C++ program to check if binary
// representations of two numbers are anagram.
#include <bits/stdc++.h>
#define ull unsigned long long int
using namespace std;

// Returns true if binary representations of
// a and b are anagram.
bool bit_anagram_check(ull a, ull b)
{
    // _popcnt64(a) gives number of 1's present
    // in binary representation of a.
    return (_popcnt64(a) == _popcnt64(b))
}

int main()
{
    ull a = 8, b = 4;
    cout << bit_anagram_check(a, b) << endl;
    return 0;
}
```

Java

```
 // An efficient Java program to check if binary
// representations of two numbers are anagram
import java.io.*;

class GFG
{
    // Function returns true if binary representations of
    // a and b are anagram
    static boolean bit_anagram_check(long a, long b)
    {
        // Long.bitCount(a) gives number of 1's present
        // in binary representation of a
        return (Long.bitCount(a) == Long.bitCount(b));
    }

    // driver program
    public static void main (String[] args)
    {
        long a = 8, b = 4;
        if(bit_anagram_check(a, b))
            System.out.println("1");
        else
            System.out.println("0");
    }
}

// Contributed by Pramod Kumar
```

Output:

```
1
```

Note that the above code uses GCC specific functions. If we wish to write code for other compilers, we may use Count set bits in an integer.

Time Complexity : O (1)
Auxiliary Space : O (1) No extra space is getting used.

Source

https://www.geeksforgeeks.org/check-binary-representations-two-numbers-anagram/

Chapter 59

Check if binary string multiple of 3 using DFA

Check if binary string multiple of 3 using DFA - GeeksforGeeks

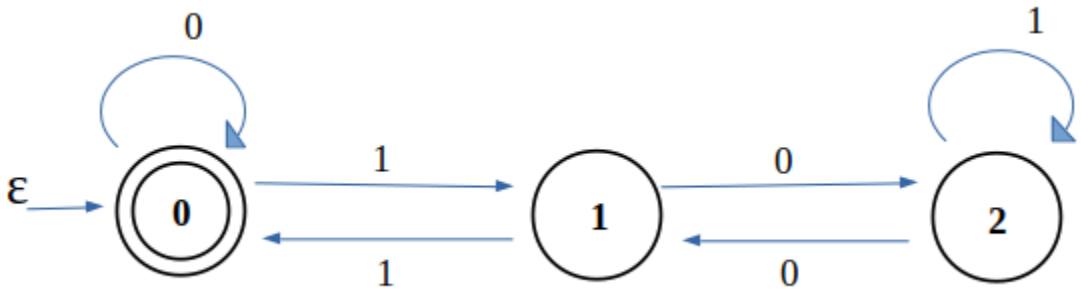
Given a string of binary characters, check if it is multiple of 3 or not.

Examples :

```
Input :  1 0 1 0
Output : NO
Explanation : (1 0 1 0) is 10 and hence
not a multiple of 3

Input :  1 1 0 0
Output : YES
Explanation : (1 1 0 0) is 12 and hence
a multiple of 3
```

Approach : One simple method is to convert the binary number into its decimal representation and then check if it is a multiple of 3 or not. Now, when it comes to **DFA (Deterministic Finite Automata)**, there is no concept of memory i.e. you cannot store the string when it is provided, so the above method would not be applicable. In simple terms, a DFA takes a string as input and process it. If it reaches final state, it is accepted, else rejected. As you cannot store the string, so input is taken character by character.

The DFA for given problem is :



As, when a number is divided by 3, there are only 3 possibilities. The remainder can be either 0, 1 or 2. Here, state 0 represents that the remainder when the number is divided by 3 is 0. State 1 represents that the remainder when the number is divided by 3 is 1 and similarly state 2 represents that the remainder when the number is divided by 3 is 2. So if a string reaches state 0 in the end, it is accepted otherwise rejected.

Below is the implementation of above approach :

C++

```
 // CPP Program to illustrate
// DFA for multiple of 3
#include <bits/stdc++.h>
using namespace std;

// checks if binary characters
// are multiple of 3
bool isMultiple3(char c[], int size)
{
    // initial state is 0th
    char state = '0';

    for (int i = 0; i < size; i++) {

        // storing binary digit
        char digit = c[i];

        switch (state) {

        // when state is 0
        case '0':
            if (digit == '1')
                state = '1';
            break;

        // when state is 1
```

```
        case '1':
            if (digit == '0')
                state = '2';
            else
                state = '0';
            break;

        // when state is 2
        case '2':
            if (digit == '0')
                state = '1';
            break;
        }
    }

    // if final state is 0th state
    if (state == '0')
        return true;
    return false;
}

// Driver's Code
int main()
{
    // size of binary array
    int size = 5;

    // array of binary numbers
    // Here it is 21 in decimal
    char c[] = { '1', '0', '1', '0', '1' };

    // if binary numbers are a multiple of 3
    if (isMultiple3(c, size))
        cout << "YES\n";
    else
        cout << "NO\n";

    return 0;
}
```

Java

```
 // Java Program to illustrate
// DFA for multiple of 3
import java.io.*;

class GFG
{
```

```
// checks if binary characters
// are multiple of 3
static boolean isMultiple3(char c[], int size)
{
    // initial state is 0th
    char state = '0';

    for (int i = 0; i < size; i++) {

        // storing binary digit
        char digit = c[i];

        switch (state) {

        // when state is 0
        case '0':
            if (digit == '1')
                state = '1';
            break;

        // when state is 1
        case '1':
            if (digit == '0')
                state = '2';
            else
                state = '0';
            break;

        // when state is 2
        case '2':
            if (digit == '0')
                state = '1';
            break;
        }
    }

    // if final state is 0th state
    if (state == '0')
        return true;
    return false;
}




// Driver Code
public static void main (String[] args)
{
```

```
        // size of binary array
        int size = 5;

        // array of binary numbers
        // Here it is 21 in decimal
        char c[] = { '1', '0', '1', '0', '1' };

        // if binary numbers are a multiple of 3
        if (isMultiple3(c, size))
            System.out.println ("YES");
        else
            System.out.println ("NO");

    }
}
// This code is contributed by vt_m
```

C#

```
 // C# Program to illustrate
// DFA for multiple of 3
using System;

class GFG {

    // checks if binary characters
    // are multiple of 3
    static bool isMultiple3(char []c, int size)
    {
        // initial state is 0th
        char state = '0';

        for (int i = 0; i < size; i++)
        {

            // storing binary digit
            char digit = c[i];

            switch (state)
            {

                // when state is 0
                case '0':
                    if (digit == '1')
                        state = '1';
                    break;

                // when state is 1
```

```
                case '1':
                    if (digit == '0')
                        state = '2';
                    else
                        state = '0';
                    break;

                // when state is 2
                case '2':
                    if (digit == '0')
                        state = '1';
                    break;
            }
        }

        // if final state is 0th state
        if (state == '0')
            return true;

        return false;
    }

    // Driver Code
    public static void Main ()
    {

        // size of binary array
        int size = 5;

        // array of binary numbers
        // Here it is 21 in decimal
        char []c = { '1', '0', '1', '0', '1' };

        // if binary numbers are a multiple of 3
        if (isMultiple3(c, size))
            Console.WriteLine ("YES");
        else
            Console.WriteLine ("NO");
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to illustrate
// DFA for multiple of 3
```

```
// checks if binary characters
// are multiple of 3
function isMultiple3($c,$size)
{

    // initial state is 0th
    $state = '0';

    for ($i = 0; $i < $size; $i++)
    {

        // storing binary digit
        $digit = $c[$i];

        switch ($state)
        {

            // when state is 0
            case '0':
                if ($digit == '1')
                    $state = '1';
                break;

            // when state is 1
            case '1':
                if ($digit == '0')
                    $state = '2';
                else
                    $state = '0';
                break;

            // when state is 2
            case '2':
                if ($digit == '0')
                    $state = '1';
                break;
        }
    }

    // if final state is 0th state
    if ($state == '0')
        return true;
    return false;
}

    // Drive Code
```

```
    // size of binary array
    $size = 5;

    // array of binary numbers
    // Here it is 21 in decimal
    $c= array('1', '0', '1', '0', '1');

    // if binary numbers are
    // a multiple of 3
    if (isMultiple3($c, $size))
        echo "YES\n";
    else
        echo "NO\n";

//This code is contributed by mits
?>
```

Output:

```
YES
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/check-binary-string-multiple-3-using-dfa/

Chapter 60

Check if bits in range L to R of two numbers are complement of each other or not

Check if bits in range L to R of two numbers are complement of each other or not - GeeksforGeeks

Given two non-negative numbers **a** and **b** and two values **l** and **r**. The problem is to check whether all bits at corresponding positions in the range **l** to **r** in both the given numbers are complement of each other or not.

The bits are numbered from right to left, i.e., the least significant bit is considered to be at first position.

Examples:

```
Input: a = 10, b = 5
       l = 1, r = 3
Output: Yes
(10)10 = (1010)2
(5)10 = (101)2 = (0101)2
All the bits in the range 1 to 3 are complement of each other.

Input: a = 21, b = 13
       l = 2, r = 4
Output: No
(21)10 = (10101)2
(13)10 = (1101)2 = (1101)2
All the bits in the range 2 to 4 are not complement of each other.
```

Approach: Below are the steps to solve the problem

- Calculate **xor_value** = a ^ b.
- Check whether all the bits are set or not in the range **l** to **r** in **xor_value**. Refer this post.

Below is the implementation of the above approach.

Source

https://www.geeksforgeeks.org/check-if-all-the-bits-of-two-numbers-that-lies-within-range-l-to-r-are-complement-o

C++

```
 // C++ implementation to check
// whether all the bits in the given range
// of two numbers are complement of each other
#include <bits/stdc++.h>
using namespace std;

// function to check whether all the bits
// are set in the given range or not
bool allBitsSetInTheGivenRange(unsigned int n,
                               unsigned int l,
                               unsigned int r)
{
    // calculating a number 'num' having 'r'
    // number of bits and bits in the range l
    // to r are the only set bits
    int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

    // new number which will only have one or more
    // set bits in the range l to r and nowhere else
    int new_num = n & num;

    // if both are equal, then all bits are set
    // in the given range
    if (num == new_num)
        return true;

    // else all bits are not set
    return false;
}

// function to check whether all the bits in the given range
// of two numbers are complement of each other
bool bitsAreComplement(unsigned int a, unsigned int b,
                       unsigned int l, unsigned int r)
```

```
{
    unsigned int xor_value = a ^ b;
    return allBitsSetInTheGivenRange(xor_value, l, r);
}

// Driver Code
int main()
{
    unsigned int a = 10, b = 5;
    unsigned int l = 1, r = 3;

    if (bitsAreComplement(a, b, l, r))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

// Java implementation to check
// whether all the bits in the
// given range of two numbers
// are complement of each other
class GFG
{
// function to check whether
// all the bits are set in
// the given range or not
static boolean allBitsSetInTheGivenRange(int n,
int l, int r)
{
// calculating a number 'num'
// having 'r' number of bits
// and bits in the range l
// to r are the only set bits
int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1); // new number which will only // have one or more set bits // in the range l to r and // nowhere else int new_num = n & num; // if both are equal, // then all bits are set // in the given range if (num == new_num) return true; // else all bits are not set return false; } // function to check whether all // the bits in the given range // of two numbers are complement // of each other static boolean bitsAreComplement(int a, int b, int l, int r) { int xor_value = a ^ b; return allBitsSetInTheGivenRange(xor_value, l, r); } // Driver Code public static void main(String []args) { int a = 10, b = 5; int l = 1, r = 3; if (bitsAreComplement(a, b, l, r)) System.out.println("Yes"); else System.out.println("No"); } } // This code is contributed by Smitha [tabbyending]

Output:

```
Yes
```

Improved By : Smitha Dinesh Semwal

Chapter 61

Check if bits of a number has count of consecutive set bits in increasing order

Check if bits of a number has count of consecutive set bits in increasing order - GeeksforGeeks

Given a integer n > 0, the task is to find whether in the bit pattern of integer count of continuous 1's are in increasing from left to right.
Examples :

```
Input:19
Output:Yes
Explanation: Bit-pattern of 19 = 10011,
Counts of continuous 1's from left to right
are 1, 2 which are in increasing order.

Input  : 183
Output : yes
Explanation: Bit-pattern of 183 = 10110111,
Counts of continuous 1's from left to right
are 1, 2, 3 which are in increasing order.
```

A **simple solution** is to store binary representation of given number into a string, then traverse from left to right and count the number of continuous 1's. For every encounter of 0 check the value of previous count of continuous 1's to that of current value, if the value of previous count is greater than the value of current count then return False, Else when string ends return True.

C++

```
 // C++ program to find if bit-pattern
// of a number has increasing value of
// continuous-1 or not.
#include<bits/stdc++.h>
using namespace std;

// Returns true if n has increasing count of
// continuous-1 else false
bool findContinuous1(int n)
{
    const int bits = 8*sizeof(int);

    // store the bit-pattern of n into
    // bit bitset- bp
    string bp = bitset <bits>(n).to_string();

    // set prev_count = 0 and curr_count = 0.
    int prev_count = 0, curr_count = 0;

    int i = 0;
    while (i < bits)
    {
        if (bp[i] == '1')
        {
            // increment current count of continuous-1
            curr_count++;
            i++;
        }

        // traverse all continuous-0
        else if (bp[i-1] == '0')
        {
            i++;
            curr_count = 0;
            continue;
        }

        // check  prev_count and curr_count
        // on encounter of first zero after
        // continuous-1s
        else
        {
            if (curr_count < prev_count)
                return 0;
            i++;
            prev_count=curr_count;
            curr_count = 0;
        }
```

```
    }

    // check for last sequence of continuous-1
    if (prev_count > curr_count && (curr_count != 0))
        return 0;

    return 1;
}

// Driver code
int main()
{
    int n = 179;
    if (findContinuous1(n))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Output :

```
Yes
```

An **efficient solution** is to use decimal to binary conversion loop that divides number by 2 and take remainder as bit. This loop finds bits from right to left. So we check if right to left is in decreasing order or not.

Below is the implementation.

C++

```
 // C++ program to check if counts of consecutive
// 1s are increasing order.
#include<bits/stdc++.h>
using namespace std;

// Returns true if n has counts of consecutive
// 1's are increasing order.
bool areSetBitsIncreasing(int n)
{
    // Initialize previous count
    int prev_count = INT_MAX;

    // We traverse bits from right to left
```

```
    // and check if counts are decreasing
    // order.
    while (n > 0)
    {
        // Ignore 0s until we reach a set bit.
        while (n > 0 && n % 2 == 0)
           n = n/2;

        // Count current set bits
        int curr_count = 1;
        while (n > 0 && n % 2 == 1)
        {
            n = n/2;
            curr_count++;
        }

        // Compare current with previous and
        // update previous.
        if (curr_count >= prev_count)
            return false;
        prev_count = curr_count;
    }

    return true;
}

// Driver code
int main()
{
    int n = 10;
    if (areSetBitsIncreasing(n))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
 // java program to check if counts of
// consecutive 1s are increasing order.
import java .io.*;

class GFG {

    // Returns true if n has counts of
    // consecutive 1's are increasing
```

```
    // order.
    static boolean areSetBitsIncreasing(int n)
    {

        // Initialize previous count
        int prev_count = Integer.MAX_VALUE;

        // We traverse bits from right to
        // left and check if counts are
        // decreasing order.
        while (n > 0)
        {

            // Ignore 0s until we reach
            // a set bit.
            while (n > 0 && n % 2 == 0)
            n = n/2;

            // Count current set bits
            int curr_count = 1;
            while (n > 0 && n % 2 == 1)
            {
                n = n/2;
                curr_count++;
            }

            // Compare current with previous
            // and update previous.
            if (curr_count >= prev_count)
                return false;
            prev_count = curr_count;
        }

        return true;
    }

    // Driver code
    static public void main (String[] args)
    {
        int n = 10;

        if (areSetBitsIncreasing(n))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

```
// This code is contributed by anuj_67.
```

Python 3

```
 # Python3 program to check if counts of
# consecutive 1s are increasing order.

import sys

# Returns true if n has counts of
# consecutive 1's are increasing order.
def areSetBitsIncreasing(n):

    # Initialize previous count
    prev_count = sys.maxsize

    # We traverse bits from right to
    # left and check if counts are
    # decreasing order.
    while (n > 0):

        # Ignore 0s until we reach a
        # set bit.
        while (n > 0 and n % 2 == 0):
            n = int(n/2)

        # Count current set bits
        curr_count = 1
        while (n > 0 and n % 2 == 1):

            n = n/2
            curr_count += 1

        # Compare current with previous
        # and update previous.
        if (curr_count >= prev_count):
            return False
        prev_count = curr_count

    return True

# Driver code
n = 10

if (areSetBitsIncreasing(n)):
    print("Yes")
else:
    print("No")
```

```
# This code is contributed by Smitha
```

C#

```
 // C# program to check if counts of
// consecutive 1s are increasing order.
using System;

class GFG {

    // Returns true if n has counts of
    // consecutive 1's are increasing
    // order.
    static bool areSetBitsIncreasing(int n)
    {

        // Initialize previous count
        int prev_count = int.MaxValue;

        // We traverse bits from right to
        // left and check if counts are
        // decreasing order.
        while (n > 0)
        {

            // Ignore 0s until we reach
            // a set bit.
            while (n > 0 && n % 2 == 0)
            n = n/2;

            // Count current set bits
            int curr_count = 1;
            while (n > 0 && n % 2 == 1)
            {
                n = n/2;
                curr_count++;
            }

            // Compare current with previous
            // and update previous.
            if (curr_count >= prev_count)
                return false;
            prev_count = curr_count;
        }

        return true;
    }
```

```
    // Driver code
    static public void Main ()
    {
        int n = 10;

        if (areSetBitsIncreasing(n))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to check if
// counts of consecutive
// 1s are increasing order.

// Returns true if n has
// counts of consecutive
// 1's are increasing order.
function areSetBitsIncreasing( $n)
{
    // Initialize previous count
    $prev_count = PHP_INT_MAX;

    // We traverse bits from right
    // to left and check if counts
    // are decreasing order.
    while ($n > 0)
    {
        // Ignore 0s until we
        // reach a set bit.
        while ($n > 0 && $n % 2 == 0)
        $n = $n / 2;

        // Count current set bits
        $curr_count = 1;
        while ($n > 0 and $n % 2 == 1)
        {
            $n = $n / 2;
            $curr_count++;
        }
```

```
        // Compare current with previous
        // and update previous.
        if ($curr_count >= $prev_count)
            return false;
        $prev_count = $curr_count;
    }

    return true;
}

// Driver code
$n = 10;
if (areSetBitsIncreasing($n))
    echo "Yes";
else
    echo "No";

// This code is contributed by anuj_67
?>
```

Output :

```
No
```

Improved By : vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/check-bits-number-count-consecutive-set-bits-increasing-order/

Chapter 62

Check if bitwise AND of any subset is power of two

Check if bitwise AND of any subset is power of two - GeeksforGeeks

Given an array **arr**[] of **n** positive integers. The task is to check if there exist any subset of the array whose bitwise AND is a power of two (i.e 1, 2, 4, 8, 16, …).

Examples:

```
Input : n = 3, arr[] = { 12, 13, 7 }
Output : Yes
Subset { 12, 7 } has Biwise AND value 4, which
is power of 2.

Input : n = 2, arr[] = { 10, 20 }
Output : No
```

Observe, for a number to be the power of 2, it should have only 1 set bit.
If n is 1, then we simply check if the number has the only single set bit.
For n is greater than one, our task becomes to choose those numbers from the array whose bitwise AND leads to an only single bit set number. To do so, we search a position, at which all elements in the set has a bit set at that position. For example, for set { 4 (100), 6 (110), 7 (111) }, at position 2 (from right to left, 0-based indexing) bit is set for all element. So, doing bitwise AND gives 4, which is a power of 2.

Below is the implementation of this approach:

C++

```
 // CPP Program to check if Bitwise AND of any
// subset is power of two
```

```
#include <bits/stdc++.h>
using namespace std;

const int NUM_BITS = 32;

// Check for power of 2 or not
bool isPowerOf2(int num)
{
    return (num && !(num & (num - 1)));
}

// Check if there exist a subset whose bitwise AND
// is power of 2.
bool checkSubsequence(int arr[], int n)
{
    // if there is only one element in the set.
    if (n == 1)
       return isPowerOf2(arr[0]);

    // Finding a number with all bit sets.
    int total = 0;
    for (int i = 0; i < NUM_BITS; i++)
        total = total | (1 << i);

    // check all the positions at which the bit is set.
    for (int i = 0; i < NUM_BITS; i++) {

        int ans = total;
        for (int j = 0; j < n; j++) {

            // include all those elements whose
            // i-th bit is set
            if (arr[j] & (1 << i))
                ans = ans & arr[j];
        }

        // check for the set contains elements
        // make a power of 2 or not
        if (isPowerOf2(ans))
            return true;
    }
    return false;
}

// Driver Program
int main()
{
    int arr[] = { 12, 13, 7 };
```

```
    int n = sizeof(arr) / sizeof(arr[0]);
    if (checkSubsequence(arr, n))
        printf("YES\n");
    else
        printf("NO\n");
    return 0;
}
```

Java

```
 // Java Program to check if Bitwise AND of any
// subset is power of two
import java.io.*;
import java.util.*;

public class GFG {

    static int NUM_BITS = 32;

    // Check for power of 2 or not
    static boolean isPowerOf2(int num)
    {
        if(num != 0 && (num & (num - 1)) == 0)
            return true;
        return false;
    }

    // Check if there exist a
    // subset whose bitwise AND
    // is power of 2.
    static boolean checkSubsequence(int []arr, int n)
    {

        // if there is only one
        // element in the set.
        if (n == 1)
            return isPowerOf2(arr[0]);

        // Finding a number with
        // all bit sets.
        int total = 0;
        for (int i = 0; i < NUM_BITS; i++)
            total = total | (1 << i);

        // check all the positions
        // at which the bit is set.
        for (int i = 0; i < NUM_BITS; i++)
        {
```

```
            int ans = total;
            for (int j = 0; j < n; j++)
            {

                // include all those
                // elements whose
                // i-th bit is set
                int p = arr[j] & (1 << i);
                if (p == 0)
                    ans = ans & arr[j];
            }

            // check for the set
            // contains elements
            // make a power of 2
            // or not
            if (isPowerOf2(ans))
                return true;
        }
        return false;
    }

    // Driver Code
    public static void main(String args[])
    {
        int []arr = {12, 13, 7};
        int n = arr.length;
        if (checkSubsequence(arr, n))
            System.out.println("YES");
        else
            System.out.println("NO");
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

Python3

```
 # Python3 Program to check if Bitwise AND of any
# subset is power of two

NUM_BITS = 32

# Check for power of 2 or not
def isPowerOf2(num):
    return (num and (num & (num - 1)) == 0)
```

```
# Check if there exist a subset whose bitwise AND
# is power of 2.
def checkSubsequence(arr, n):

    # if there is only one element in the set.
    if (n == 1):
        return isPowerOf2(arr[0])

    # Finding a number with all bit sets.
    total = 0
    for i in range(0, NUM_BITS):
        total = total | (1 << i)

    # check all the positions at which the bit is set.
    for i in range(0, NUM_BITS):

        ans = total
        for j in range(0, n):

            # include all those elements whose
            # i-th bit is set
            if (arr[j] & (1 << i)):
                ans = ans & arr[j]

        # check for the set contains elements
        # make a power of 2 or not
        if (isPowerOf2(ans)):
            return True
    return False

# Driver Program
arr = [ 12, 13, 7 ]
n = len(arr)
if (checkSubsequence(arr, n)):
    print ("YES\n")
else:
    print ("NO\n")

# This code is contributed by Manish Shaw
# (manishshaw1)
```

C#

```
 // C# Program to check if Bitwise AND of any
// subset is power of two
using System;
using System.Collections.Generic;
```

```
class GFG {

    static int NUM_BITS = 32;

    // Check for power of 2 or not
    static bool isPowerOf2(int num)
    {
        if(num != 0 && (num & (num - 1)) == 0)
            return true;
        return false;
    }

    // Check if there exist a
    // subset whose bitwise AND
    // is power of 2.
    static bool checkSubsequence(int []arr, int n)
    {

        // if there is only one
        // element in the set.
        if (n == 1)
            return isPowerOf2(arr[0]);

        // Finding a number with
        // all bit sets.
        int total = 0;
        for (int i = 0; i < NUM_BITS; i++)
            total = total | (1 << i);

        // check all the positions
        // at which the bit is set.
        for (int i = 0; i < NUM_BITS; i++)
        {

            int ans = total;
            for (int j = 0; j < n; j++)
            {

                // include all those
                // elements whose
                // i-th bit is set
                int p = arr[j] & (1 << i);
                if (p == 0)
                    ans = ans & arr[j];
            }

            // check for the set
```

```
            // contains elements
            // make a power of 2
            // or not
            if (isPowerOf2(ans))
                return true;
        }
        return false;
    }

    // Driver Code
    public static void Main()
    {
        int []arr = {12, 13, 7};
        int n = arr.Length;
        if (checkSubsequence(arr, n))
            Console.Write("YES\n");
        else
            Console.Write("NO\n");
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

PHP

```
 <?php
// PHP Program to check if
// Bitwise AND of any subset
// is power of two

// Check for power of 2 or not
function isPowerOf2($num)
{
    return ($num && !($num & ($num - 1)));
}

// Check if there exist a
// subset whose bitwise AND
// is power of 2.
function checkSubsequence($arr, $n)
{
    $NUM_BITS = 32;

    // if there is only one
    // element in the set.
    if ($n == 1)
    return isPowerOf2($arr[0]);
```

```
    // Finding a number with
    // all bit sets.
    $total = 0;
    for($i = 0; $i < $NUM_BITS; $i++)
        $total = $total | (1 << $i);

    // check all the positions at
    // which the bit is set.
    for($i = 0; $i < $NUM_BITS; $i++)
    {

        $ans = $total;
        for ($j = 0; $j < $n; $j++)
        {

            // include all those
            // elements whose
            // i-th bit is set
            if ($arr[$j] & (1 << $i))
                $ans = $ans & $arr[$j];
        }

        // check for the set
        // contains elements
        // make a power of 2 or not
        if (isPowerOf2($ans))
            return true;
    }
    return false;
}

    // Driver Code
    $arr= array(12, 13, 7);
    $n = sizeof($arr) / sizeof($arr[0]);
    if (checkSubsequence($arr, $n))
        echo "YES";
    else
        echo "NO";

// This code is contributed by mits
?>
```

Output:

```
YES
```

Reference:
https://stackoverflow.com/questions/35990794/subset-of-array-a-in-which-if-we-do-and-of-all-elements-of-that-sub

Improved By : Mithun Kumar, manishshaw1

Source

https://www.geeksforgeeks.org/check-bitwise-subset-power-two/

Chapter 63

Check if concatenation of two strings is balanced or not

Check if concatenation of two strings is balanced or not - GeeksforGeeks

Given two bracket sequences S1 and S2 consisting of '(' and ')'. The task is to check if the string obtained by concatenating both the sequences is balanced or not. Concatenation can be done by s1+s2 or s2+s1.

Examples:

> **Input:** s1 = ")()(()))", s2 = "(()(()("
> **Output:** Balanced
> s2 + s1 = "(()(()()()(()))", which
> is a balanced paranthesis sequence.
>
> **Input:** s1 = "(()))(", s2 = "())(())"
> **Output:** Not balanced
> s1 + s2 = "(()))(())(())" –> Not balanced
> s2 + s1 = "())(())(()))(" –> Not balanced

A **naive** solution is to first concatenate both sequences and then check if the resultant sequence is balanced or not using a stack. First, check if s1 + s2 is balanced or not. If not, then check if s2 + s1 is balanced or not. To check if a given sequence of brackets is balanced or not using a stack, the following algorithm can be used.

1. Declare a character stack S.
2. Now traverse the expression string exp.
    - If the current character is a starting bracket ('(' or '{' or '[') then push it to stack.
    - If the current character is a closing bracket (')' or '}' or ']') then pop from the stack and if the popped character is the matching starting bracket then fine else parenthesis are not balanced.

3. After complete traversal, if there is some starting bracket left in stack then "not balanced".

Below is the implementation of above approach:

C++

```
 // CPP program to check if sequence obtained
// by concatenating two bracket sequences
// is balanced or not.
#include <bits/stdc++.h>
using namespace std;

// Check if given string is balanced bracket
// sequence or not.
bool isBalanced(string s)
{

    stack<char> st;

    int n = s.length();

    for (int i = 0; i < n; i++) {

        // If current bracket is an opening
        // bracket push it to stack.
        if (s[i] == '(')
            st.push(s[i]);

        // If current bracket is a closing
        // bracket then pop from stack if
        // it is not empty. If stack is empty
        // then sequence is not balanced.
        else {
            if (st.empty()) {
                return false;
            }

            else
                st.pop();
        }
    }

    // If stack is not empty, then sequence
    // is not balanced.
    if (!st.empty())
        return false;
```

```
    return true;
}

// Function to check if string obtained by
// concatenating two bracket sequences is
// balanced or not.
bool isBalancedSeq(string s1, string s2)
{

    // Check if s1 + s2 is balanced or not.
    if (isBalanced(s1 + s2))
        return true;

    // Check if s2 + s1 is balanced or not.
    return isBalanced(s2 + s1);
}

// Driver code.
int main()
{
    string s1 = ")()(()))";
    string s2 = "(()(()(";

    if (isBalancedSeq(s1, s2))
        cout << "Balanced";
    else
        cout << "Not Balanced";

    return 0;
}
```

Output:

```
Balanced
```

Time complexity: O(n)
Auxiliary Space: O(n)

An **efficient** solution is to check if given sequences can result in balanced parenthesis sequence without using a stack, i.e., in constant extra space.
Let the concatenated sequence is s. There are two possibilities: either s = s1 + s2 is balanced or s = s2 + s1 is balanced. Check for both possibilities whether s is balanced or not.

- If s is balanced, then the number of opening brackets in s should always be greater than or equal to the number of closing brackets in S at any instant of traversing it. This is because if at any instant number of closing brackets in s is greater than the

number of opening brackets, then the last closing bracket will not have a matching opening bracket (that is why the count is more) in s.
- If the sequence is balanced then at the end of traversal, the number of opening brackets in s is equal to the number of closing brackets in s.

Below is the implementation of above approach:

C++

```
 // C++ program to check if sequence obtained
// by concatenating two bracket sequences
// is balanced or not.
#include <bits/stdc++.h>
using namespace std;

// Check if given string is balanced bracket
// sequence or not.
bool isBalanced(string s)
{

    // To store result of comparison of
    // count of opening brackets and
    // closing brackets.
    int cnt = 0;

    int n = s.length();

    for (int i = 0; i < n; i++) {

        // If current bracket is an
        // opening bracket, then
        // increment count.
        if (s[i] == '('
            cnt++;

        // If current bracket is a
        // closing bracket, then
        // decrement count and check
        // if count is negative.
        else {
            cnt--;
            if (cnt < 0)
                return false;
        }
    }

    // If count is positive then
```

```
    // some opening brackets are
    // not balanced.
    if (cnt > 0)
        return false;

    return true;
}

// Function to check if string obtained by
// concatenating two bracket sequences is
// balanced or not.
bool isBalancedSeq(string s1, string s2)
{

    // Check if s1 + s2 is balanced or not.
    if (isBalanced(s1 + s2))
        return true;

    // Check if s2 + s1 is balanced or not.
    return isBalanced(s2 + s1);
}

// Driver code.
int main()
{
    string s1 = ")()(()))";
    string s2 = "(()(()(";

    if (isBalancedSeq(s1, s2))
        cout << "Balanced";
    else
        cout << "Not Balanced";

    return 0;
}
```

Java

```
 // Java program to check if
// sequence obtained by
// concatenating two bracket
// sequences is balanced or not.
import java.io.*;

class GFG
{

// Check if given string
```

```
// is balanced bracket
// sequence or not.
static boolean isBalanced(String s)
{

// To store result of comparison
// of count of opening brackets
// and closing brackets.
int cnt = 0;
int n = s.length();
for (int i = 0; i < n; i++)
{

    // If current bracket is
    // an opening bracket,
    // then increment count.
    if (s.charAt(i) =='(')
    {
        cnt = cnt + 1;
    }

    // If current bracket is a
    // closing bracket, then
    // decrement count and check
    // if count is negative.
    else
    {
        cnt = cnt - 1;
        if (cnt < 0)
            return false;
    }
}

// If count is positive then
// some opening brackets are
// not balanced.
if (cnt > 0)
    return false;

return true;
}

// Function to check if string
// obtained by concatenating
// two bracket sequences is
// balanced or not.
static boolean isBalancedSeq(String s1,
                             String s2)
```

```
{

// Check if s1 + s2 is
// balanced or not.
if (isBalanced(s1 + s2))
    return true;

// Check if s2 + s1 is
// balanced or not.
return isBalanced(s2 + s1);
}

// Driver code
public static void main(String [] args)
{
    String s1 = ")()(()))";
    String s2 = "(()(()(";

    if (isBalancedSeq(s1, s2))
    {
        System.out.println("Balanced");
    }
    else
    {
        System.out.println("Not Balanced");
    }
}
}

// This code is contributed
// by Shivi_Aggarwal
```

Python3

```
 # Python3 program to check
# if sequence obtained by
# concatenating two bracket
# sequences is balanced or not.

# Check if given string
# is balanced bracket
# sequence or not.
def isBalanced(s):

    # To store result of
    # comparison of count
    # of opening brackets
    # and closing brackets.
```

```
    cnt = 0
    n = len(s)

    for i in range(0, n):
        if (s[i] == '('):
            cnt = cnt + 1
        else :
            cnt = cnt - 1
            if (cnt < 0):
                return False
    if (cnt > 0):
        return False

    return True

def isBalancedSeq(s1, s2):

    if (isBalanced(s1 + s2)):
        return True

    return isBalanced(s2 + s1)

# Driver code
a = ")()(()))))";
b = "(()(()(";

if (isBalancedSeq(a, b)):
        print("Balanced")
else:
    print("Not Balanced")

# This code is contributed
# by Shivi_Aggarwal
```

Output:

```
Balanced
```

Time complexity: O(n)
Auxiliary Space: O(1)

Improved By : Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/check-if-concatenation-of-two-strings-is-balanced-or-not/

Chapter 64

Check if given four integers (or sides) make rectangle

Check if given four integers (or sides) make rectangle - GeeksforGeeks

Given four positive integers, determine if there's a rectangle such that the lengths of its sides are a, b, c and d (in any order).

Examples :

```
Input : 1 1 2 2
Output : Yes

Input : 1 2 3 4
Output : No
```

Approach 1 :- We will check, if any of the two integers are equal and make sure rest of two are also equal using few if else conditions.

C++

```
 // A simple program to find if given 4
// values can represent 4 sides of rectangle
#include <iostream>
using namespace std;

// Function to check if the given
// integers value make a rectangle
bool isRectangle(int a, int b, int c, int d)
{
    // Square is also a rectangle
```

```
    if (a == b == c == d)
        return true;

    else if (a == b && c == d)
        return true;
    else if (a == d && c == b)
        return true;
    else if (a == c && d == b)
        return true;
    else
        return false;
}

// Driver code
int main()
{
    int a, b, c, d;
    a = 1, b = 2, c = 3, d = 4;
    if (isRectangle(a, b, c, d))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // A simple program to find if
// given 4 values can represent
// 4 sides of rectangle
class GFG {

    // Function to check if the given
    // integers value make a rectangle
    static boolean isRectangle(int a, int b,
                               int c, int d)
    {
        // Square is also a rectangle
        if (a == b && a == c &&
            a == d && c == d &&
            b == c && b == d)
            return true;

        else if (a == b && c == d)
            return true;
        else if (a == d && c == b)
            return true;
        else if (a == c && d == b)
```

```
            return true;
        else
            return false;
    }

    // Driver code
    public static void main(String[] args)
    {

        int a = 1, b = 2, c = 3, d = 4;
        if (isRectangle(a, b, c, d))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

// This code is contributed by prerna saini.
```

Python3

```
 # A simple program to find if given 4
# values can represent 4 sides of rectangle

# Function to check if the given
# integers value make a rectangle
def isRectangle(a, b, c, d):

    # Square is also a rectangle
    if a == b == c == d:
        return True

    elif a == b and c == d:
        return True

    elif a == d and c == b:
        return True

    elif a == c and d == b:
        return True

    return False


# Driver code
a, b, c, d = 1, 2, 3, 4
print("Yes" if isRectangle(a, b, c, d) else "No")
```

```
# This code is contributed by Ansu Kumari.
```

C#

```
 // A simple program to find if
// given 4 values can represent
// 4 sides of rectangle
using System;

class GFG {

    // Function to check if the given
    // integers value make a rectangle
    static bool isRectangle(int a, int b,
                            int c, int d)
    {
        // Square is also a rectangle
        if (a == b && a == c && a == d &&
            c == d && b == c && b == d)
            return true;

        else if (a == b && c == d)
            return true;

        else if (a == d && c == b)
            return true;

        else if (a == c && d == b)
            return true;

        else
            return false;
    }

    // Driver code
    public static void Main()
    {

        int a = 1, b = 2, c = 3, d = 4;
        if (isRectangle(a, b, c, d))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by vt_m.
```

Output :

```
No
```

Approach 2 :- XORing the given lengths, will given value 0.

C++

```
 // An efficient program to find if given 4
// values can represent 4 sides of rectangle
#include <iostream>
using namespace std;

// Function to check if the given
// integers value make a rectangle
bool isRectangle(int a, int b, int c, int d)
{
    if (a ^ b ^ c ^ d)
        return false;
    else
        return true;
}

// Driver code
int main()
{
    int a, b, c, d;
    a = 3, b = 2, c = 3, d = 2;
    if (isRectangle(a, b, c, d))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // An efficient Java program to find if given 4
// values can represent 4 sides of rectangle

class GFG
{
    // Function to check if the given
    // integers value make a rectangle
    static boolean isRectangle(int a, int b,
```

```
                            int c, int d)
    {
        if ((a ^ b ^ c ^ d) != 0)
            return false;
        else
            return true;
    }

    // Driver code
    public static void main(String[] args)
    {
        int a, b, c, d;
        a = 3; b = 2; c = 3; d = 2;
        if (isRectangle(a, b, c, d))
            System.out.println("Yes");
        else
            System.out.println("No");

    }
}

// This code is contributed by
// Smitha Dinesh Semwal
```

Python3

```
 # An efficient program to find if given 4
# values can represent 4 sides of rectangle

# Function to check if the given
# integers value make a rectangle
def isRectangle(a, b, c, d):

    if a ^ b ^ c ^ d:
        return False

    return True


# Driver code
a, b, c, d = 3, 2, 3, 2
print("Yes"    if isRectangle(a, b, c, d) else "No")


# This code is contributed by Ansu Kumari.
```

C#

```
 // An efficient C# program to find
// if given 4 values can represent
// 4 sides of rectangle
using System;

class GFG {

    // Function to check if the given
    // integers value make a rectangle
    static bool isRectangle(int a, int b,
                            int c, int d)
    {
        if ((a ^ b ^ c ^ d) != 0)
            return false;
        else
            return true;
    }

    // Driver code
    public static void Main()
    {
        int a, b, c, d;
        a = 3;
        b = 2;
        c = 3;
        d = 2;

        if (isRectangle(a, b, c, d))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by vt_m
```

Output :

```
Yes
```

Improved By : Aditya raj 9

Source

https://www.geeksforgeeks.org/check-given-four-integers-sides-make-rectangle/

Chapter 65

Check if given number is a power of d where d is a power of 2

Check if given number is a power of d where d is a power of 2 - GeeksforGeeks

Given an integer n, find whether it is a power of d or not, where d is itself a power of 2.

Examples:

```
Input : n = 256, d = 16
Output : Yes

Input : n = 32, d = 16
Output : No
```

Method 1 Take log of the given number on base d, and if we get an integer then number is power of d.

Method 2 Keep dividing the number by d, i.e, do n = n/d iteratively. In any iteration, if n%d becomes non-zero and n is not 1 then n is not a power of d, otherwise n is a power of d.

Method 3(Bitwise)
A number n is a power of d if following conditions are met.
a) There is only one bit set in the binary representation of n (Note : d is a power of 2)
b) The count of zero bits before the (only) set bit is a multiple of $\log_2(d)$.

For example: For n = 16 (10000) and d = 4, 16 is a power of 4 because there is only one bit set and count of 0s before the set bit is 4 which is a multiple of $\log_2(4)$.

C++

```
 // CPP program to find if a number is power
// of d where d is power of 2.
#include<stdio.h>

unsigned int Log2n(unsigned int n)
{
return (n > 1)? 1 + Log2n(n/2): 0;
}

bool isPowerOfd(unsigned int n, unsigned int d)
{
int count = 0;

/* Check if there is only one bit set in n*/
if (n && !(n&(n-1)) )
{
    /* count 0 bits before set bit */
    while (n > 1)
    {
    n >>= 1;
    count += 1;
    }

    /* If count is a multiple of log2(d)
    then return true else false*/
    return (count%(Log2n(d)) == 0);
}

/* If there are more than 1 bit set
    then n is not a power of 4*/
return false;
}

/* Driver program to test above function*/
int main()
{
int n = 64, d = 8;
if (isPowerOfd(n, d))
    printf("%d is a power of %d", n, d);
else
    printf("%d is not a power of %d", n, d);
return 0;
}
```

Java

```
 // Java program to find if
// a number is power of d
```

```
// where d is power of 2.

class GFG
{
static int Log2n(int n)
{
    return (n > 1)? 1 +
    Log2n(n / 2): 0;
}

static boolean isPowerOfd(int n,
                          int d)
{
int count = 0;

/* Check if there is
only one bit set in n*/
if (n > 0 && (n &
(n - 1)) == 0)
{
    /* count 0 bits
    before set bit */
    while (n > 1)
    {
        n >>= 1;
        count += 1;
    }

    /* If count is a multiple
    of log2(d) then return
    true else false*/
    return (count %
        (Log2n(d)) == 0);
}

/* If there are more
than 1 bit set then
n is not a power of 4*/
return false;
}

// Driver Code
public static void main(String[] args)
{
    int n = 64, d = 8;
    if (isPowerOfd(n, d))
        System.out.println(n +
                    " is a power of " + d);
```

```
    else
        System.out.println(n +
                    " is not a power of " + d);
}
}

// This code is contributed by mits
```

Python3

```
 # Python3 program to find if a number
# is power of d where d is power of 2.

def Log2n(n):
    return (1 + Log2n(n / 2)) if (n > 1) else 0;

def isPowerOfd(n, d):
    count = 0;

    # Check if there is only
    # one bit set in n

    if (n and (n & (n - 1))==0):
        # count 0 bits
        # before set bit
        while (n > 1):
            n >>= 1;
            count += 1;
        # If count is a multiple of log2(d)
        # then return true else false
        return (count%(Log2n(d)) == 0);

    # If there are more than 1 bit set
    # then n is not a power of 4
    return False;

# Driver Code
n = 64;
d = 8;
if (isPowerOfd(n, d)):
    print(n,"is a power of",d);
else:
    print(n,"is not a power of",d);

# This code is contributed by mits
```

C#

```
 // C# program to find if
// a number is power of d
// where d is power of 2.
using System;

class GFG
{
static int Log2n(int n)
{
    return (n > 1)? 1 +
    Log2n(n / 2): 0;
}

static bool isPowerOfd(int n,
                    int d)
{
int count = 0;

/* Check if there is
only one bit set in n*/
if (n > 0 && (n & (n - 1)) == 0)
{
    /* count 0 bits
    before set bit */
    while (n > 1)
    {
    n >>= 1;
    count += 1;
    }

    /* If count is a multiple
    of log2(d) then return
    true else false*/
    return (count % (Log2n(d)) == 0);
}

/* If there are more than
1 bit set then n is not
a power of 4*/
return false;
}

// Driver Code
static void Main()
{
int n = 64, d = 8;
if (isPowerOfd(n, d))
    Console.WriteLine("{0} is a " +
```

```
                    "power of {1}",
                            n, d);
else
    Console.WriteLine("{0} is not a"+
                    " power of {1}",
                            n, d);
}

// This code is contributed by mits
}
```

PHP

```
 <?php
// PHP program to find if a number
// is power of d where d is power of 2.

function Log2n($n)
{
    return ($n > 1)? 1 +
    Log2n($n / 2): 0;
}

function isPowerOfd($n, $d)
{
    $count = 0;

// Check if there is only
// one bit set in n
if ($n && !($n & ($n - 1)))
{

    // count 0 bits
    // before set bit
    while ($n > 1)
    {
        $n >>= 1;
        $count += 1;
    }

    /* If count is a multiple of log2(d)
    then return true else false*/
    return ($count%(Log2n($d)) == 0);
}

    /* If there are more than 1 bit set
    then n is not a power of 4*/
    return false;
```

```
}

// Driver Code
$n = 64;
$d = 8;
if (isPowerOfd($n, $d))
    echo $n," ","is a power of ", $d;
else
    echo $n," ","is not a power of ", $d;

// This code is contributed by m_kit
?>
```

Output:

```
64 is a power of 8
```

Improved By : jit_t, Mithun Kumar

Source

https://www.geeksforgeeks.org/check-given-number-power-d-d-power-2/

Chapter 66

Check if n is divisible by power of 2 without using arithmetic operators

Check if n is divisible by power of 2 without using arithmetic operators - GeeksforGeeks

Given two positive integers **n** and **m**. The problem is to check whether **n** is divisible by $\mathbf{2^m}$ or not without using arithmetic operators.

Examples:

```
Input : n = 8, m = 2
Output : Yes

Input : n = 14, m = 3
Output : No
```

Approach: If a number is divisible by 2 then it has its least significant bit (LSB) set to 0, if divisible by 4 then two LSB's set to 0, if by 8 then three LSB's set to 0 and so on. Keeping this in mind, a number **n** is divisible by $\mathbf{2^m}$ if $\mathbf{(n \ \& \ ((1 << m) - 1))}$ is equal to 0 else not.

C++

```
 // C++ implementation to chech whether n
// is divisible by pow(2, m)
#include <bits/stdc++.h>

using namespace std;

// function to chech whether n
```

```
// is divisible by pow(2, m)
bool isDivBy2PowerM(unsigned int n,
                    unsigned int m)
{
    // if expression results to 0, then
    // n is divisible by pow(2, m)
    if ((n & ((1 << m) - 1)) == 0)
        return true;

    // n is not divisible
    return false;
}

// Driver program to test above
int main()
{
    unsigned int n = 8, m = 2;
    if (isDivBy2PowerM(n, m))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // JAVA Code for Check if n is divisible
// by power of 2 without using arithmetic
// operators
import java.util.*;

class GFG {

    // function to chech whether n
    // is divisible by pow(2, m)
    static boolean isDivBy2PowerM(int n,
                                  int m)
    {
        // if expression results to 0, then
        // n is divisible by pow(2, m)
        if ((n & ((1 << m) - 1)) == 0)
            return true;

        // n is not divisible
        return false;
    }

    /* Driver program to test above function */
```

```
    public static void main(String[] args)
    {
            int n = 8, m = 2;

            if (isDivBy2PowerM(n, m))
                System.out.println("Yes");
            else
                System.out.println("No");
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python3 implementation to chech
# whether n is divisible by pow(2, m)

# function to chech whether n
# is divisible by pow(2, m)
def isDivBy2PowerM (n, m):

    # if expression results to 0, then
    # n is divisible by pow(2, m)
    if (n & ((1 << m) - 1)) == 0:
        return True

    # n is not divisible
    return False

# Driver program to test above
n = 8
m = 2
if isDivBy2PowerM(n, m):
    print("Yes")
else:
    print( "No")

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# Code for Check if n is divisible
// by power of 2 without using arithmetic
// operators
using System;
```

```
class GFG {

    // function to chech whether n
    // is divisible by pow(2, m)
    static bool isDivBy2PowerM(int n, int m)
    {
        // if expression results to 0, then
        // n is divisible by pow(2, m)
        if ((n & ((1 << m) - 1)) == 0)
            return true;

        // n is not divisible
        return false;
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int n = 8, m = 2;

        if (isDivBy2PowerM(n, m))
            Console.Write("Yes");
        else
            Console.Write("No");
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP implementation to chech whether
// n is divisible by pow(2, m)

// function to chech whether n
// is divisible by pow(2, m)
function isDivBy2PowerM($n, $m)
{

    // if expression results to 0, then
    // n is divisible by pow(2, m)
    if (($n & ((1 << $m) - 1)) == 0)
        return true;

    // n is not divisible
    return false;
}
```

```
    // Driver Code
    $n = 8;
    $m = 2;
    if (isDivBy2PowerM($n, $m))
        echo "Yes";
    else
        echo "No";

// This code is contributed by ajit
?>
```

Output:

```
Yes
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/check-n-divisible-power-2-without-using-arithmetic-operators/

Chapter 67

Check if one of the numbers is one's complement of the other

Check if one of the numbers is one's complement of the other - GeeksforGeeks

Given two non-negative integers **a** and **b**. The problem is to check if one of the two numbers is 1's complement of the other.
The **ones' complement** of a binary number is defined as the value obtained by inverting all the bits in the binary representation of the number (swapping 0s for 1s and vice versa).

Examples:

```
Input : a = 10, b = 5
Output : Yes
(10)10 = (1010)2
1's complement of 10 is
= (0101)2 = (101)2 = (5)10

Input : a = 1, b = 14
Output : Yes
(14)10 = (1110)2
1's complement of 14 is
= (0001)2 = (1)2 = (1)10
```

Approach: Following are the steps:

1. Calculate **n** = a ^ b.
2. Check whether all bits are set in the binary representation of **n**. Refer this post.

CPP

```
 // C++ implementation to check if one of the two
// numbers is one's complement of the other
#include <bits/stdc++.h>
using namespace std;

// function to check if all the bits are set
// or not in the binary representation of 'n'
bool areAllBitsSet(unsigned int n)
{
    // all bits are not set
    if (n == 0)
        return false;

    // if true, then all bits are set
    if (((n + 1) & n) == 0)
        return true;

    // else all bits are not set
    return false;
}

// function to check if one of the two numbers
// is one's complement of the other
bool isOnesComplementOfOther(unsigned int a,
                             unsigned int b)
{
    return areAllBitsSet(a ^ b);
}

// Driver program to test above
int main()
{
    unsigned int a = 10, b = 5;

    if (isOnesComplementOfOther(a,b))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
 // Java implementation to
// check if one of the two
// numbers is one's complement
// of the other
```

```
import java.util.*;
import java.lang.*;

public class GfG{

    // function to check
    // if all the bits are set
    // or not in the binary
    // representation of 'n'
    public static boolean areAllBitsSet(long n)
    {
        // all bits are not set
        if (n == 0)
            return false;

        // if true, then all bits are set
        if (((n + 1) & n) == 0)
            return true;

        // else all bits are not set
        return false;
    }

    // function to check if
    // one of the two numbers
    // is one's complement
    // of the other
    public static boolean isOnesComplementOfOther(long a,
                            long b)
    {
        return areAllBitsSet(a ^ b);
    }

    // Driver function
    public static void main(String argc[]){
        long a = 10, b = 5;

        if (isOnesComplementOfOther(a,b))
            System.out.println("Yes");
        else
            System.out.println("No");
    }

}

// This code is contributed by Sagar Shukla
```

Python3

```
 # Python3 implementation to
# check if one of the two
# numbers is one's complement
# of the other


# function to check if
# all the bits are set
# or not in the binary
# representation of 'n'
def areAllBitsSet(n):

    # all bits are not set
    if (n == 0):
        return False;

    # if True, then all bits are set
    if (((n + 1) & n) == 0):
        return True;

    # else all bits are not set
    return False;


# function to check if one
# of the two numbers is
# one's complement of the other
def isOnesComplementOfOther(a, b):

    return areAllBitsSet(a ^ b)


# Driver program
a = 1
b = 14
if (isOnesComplementOfOther(a, b)):
    print ("Yes")
else:
    print ("No")

# This code is contributed by
# Saloni Gupta 4
```

C#

```
 // C# implementation to check
```

```
// if one of the two numbers is
// one's complement of the other
using System;

class GFG {

    // function to check
    // if all the bits are set
    // or not in the binary
    // representation of 'n'
    public static bool areAllBitsSet(long n)
    {
        // all bits are not set
        if (n == 0)
            return false;

        // if true, then all bits are set
        if (((n + 1) & n) == 0)
            return true;

        // else all bits are not set
        return false;
    }

    // function to check if
    // one of the two numbers
    // is one's complement
    // of the other
    public static bool isOnesComplementOfOther(long a,
                                               long b)
    {
        return areAllBitsSet(a ^ b);
    }

    // Driver function
    public static void Main()
    {
        long a = 10, b = 5;

        if (isOnesComplementOfOther(a, b))
            Console.Write("Yes");
        else
            Console.Write("No");
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP implementation to
// check if one of the two
// numbers is one's complement
// of the other

// function to check if
// all the bits are set
// or not in the binary
// representation of 'n'
function areAllBitsSet($n)
{

    // all bits are not set
    if ($n == 0)
        return false;

    // if true, then all
    // bits are set
    if ((($n + 1) & $n) == 0)
        return true;

    // else all bits
    // are not set
    return false;
}

// function to check if
// one of the two numbers
// is one's complement of
// the other
function isOnesComplementOfOther($a,
                                 $b)
{
    return areAllBitsSet($a ^ $b);
}

    // Driver Code
    $a = 10; $b = 5;

    if (isOnesComplementOfOther($a, $b))
        echo "Yes";
    else
        echo "No";

// This code is contributed by anuj_67.
```

```
?>
```

Output:

```
Yes
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/check-one-numbers-ones-complement/

Chapter 68

Check if two numbers are bit rotations of each other or not

Check if two numbers are bit rotations of each other or not - GeeksforGeeks

Given two positive integers x and y, check if one integer is obtained by rotating bits of other.

```
Input constraint: 0 < x, y < 2^32
```

Bit Rotation: A rotation (or circular shift) is an operation similar to shift except that the bits that fall off at one end are put back to the other end.

More information on bit rotation can be found here

Example 1 :

```
Input : a = 8, b = 1
Output : yes

Explanation :
Represntation of a = 8 : 0000 0000 0000 0000 0000 0000 0000 1000
Represntation of b = 1 : 0000 0000 0000 0000 0000 0000 0000 0001
If we rotate a by 3 units right we get b, hence answer is yes
```

Example 2 :

```
Input : a = 122, b = 2147483678
Output : yes

Explanation :
```

```
Represntation of a = 122        : 0000 0000 0000 0000 0000 0000 0111 1010
Represntation of b = 2147483678 : 1000 0000 0000 0000 0000 0000 0001 1110
If we rotate a by 2 units right we get b, hence answer is yes
```

Since total bits in which x or y can be represented is 32 since x, y > 0 and x, y $< 2\hat{}32$.
So we need to find all 32 possible rotations of x and compare it with y till x and y are not equal.
To do this we use a temporary variable x64 with 64 bits which is result of concatenation of x to x ie..
x64 has first 32 bits same as bits of x and last 32 bits are also same as bits of x64.

Then we keep on shifting x64 by 1 on right side and compare the rightmost 32 bits of x64 with y.
In this way we'll be able to get all the possible bits combination due to rotation.

Here is implementation of above algorithm.

C++

```
 // C++ program to check if two numbers are bit rotations
// of each other.
#include <iostream>
using namespace std;

// function to check if  two numbers are equal
// after bit rotation
bool isRotation(unsigned int x, unsigned int y)
{
    // x64 has concatenation of x with itself.
    unsigned long long int x64 = x | ((unsigned long long int)x << 32);

    while (x64 >= y)
    {
        // comapring only last 32 bits
        if (unsigned(x64) == y)
            return true;

        // right shift by 1 unit
        x64 >>= 1;
    }
    return false;
}

// driver code to test above function
int main()
{
    unsigned int x = 122;
    unsigned int y = 2147483678;
```

```
    if (isRotation(x, y))
        cout << "yes" << endl;
    else
        cout << "no" << endl;

    return 0;
}
```

PHP

```
 <?php
// PHP program to check if two
// numbers are bit rotations of
// each other.

// function to check if two
// numbers are equal after
// bit rotation
function isRotation($x, $y)
{
    // x64 has concatenation
    // of x with itself.
    $x64 = $x | ($x << 32);

    while ($x64 >= $y)
    {
        // comapring only last 32 bits
        if (($x64) == $y)
            return 1;

        // right shift by 1 unit
        $x64 >>= 1;
    }
    return -1;
}

// Driver Code
$x = 122;
$y = 2147483678;

if (isRotation($x, $y))
    echo "yes" ,"\n";
else
    echo "no" ,"\n";

// This code is contributed by aj_36
?>
```

Output :

```
yes
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/check-two-numbers-bit-rotations-not/

Chapter 69

Check if two numbers are equal without using arithmetic and comparison operators

Check if two numbers are equal without using arithmetic and comparison operators - GeeksforGeeks

Following are not allowed to use
1) Arithmetic and Comparison Operators
2) String functions

The idea is to use XOR operator. XOR of two numbers is 0 if the numbers are same, otherwise non-zero.

C++

```
 // C++ program to check if two numbers
// are equal without using arithmetic
// and comparison operators
#include <iostream>
using namespace std;

// Function to check if two
// numbers are equal using
// XOR operator
void areSame(int a, int b)
{
   if (a^b)
   cout << "Not Same";
   else
   cout << "Same";
```

```
}

// Driver Code
int main()
{

    // Calling function
    areSame(10, 20);
}
```

Java

```
 // Java program to check if two numbers
// are equal without using arithmetic
// and comparison operators
class GFG {

// Function to check if two
// numbers are equal using
// XOR operator
static void areSame(int a, int b)
{
    if ((a ^ b) != 0)
    System.out.print("Not Same");
    else
        System.out.print("Same");
}

// Driver Code
public static void main(String[] args)
{

    // Calling function
    areSame(10, 20);
}
}

// This code is contributed by Smitha
```

Python 3

```
 # Python program to check if two numbers
# are equal without using arithmetic
# and comparison operators

def areSame(a, b):
```

```
# Function to check if two
# numbers are equal using
# XOR operator
 if ((a ^ b) != 0):
    print("Not Same")
 else:
    print("Same")

# Driver Code

areSame(10, 20)

# This code is contributed by Smitha
```

C#

```
 // C# program to check if two numbers
// are equal without using arithmetic
// and comparison operators
using System;

class GFG {

// Function to check if two
// numbers are equal using
// XOR operator
static void areSame(int a, int b)
{
    if ((a ^ b) != 0)
    Console.Write("Not Same");
    else
    Console.Write("Same");
}

// Driver Code
public static void Main(String[] args)
{

    // Calling function
    areSame(10, 20);
}
}

// This code is contributed by Smitha
```

PHP

```
 <?php
```

```
// PHP program to check if
// two numbers are equal
// without using arithmetic
// and comparison operators

// Function to check if two
// numbers are equal using
// XOR operator
function areSame($a, $b)
{
if ($a ^ $b)
echo "Not Same";
else
echo "Same";
}

// Driver Code

// Calling function
areSame(10, 20);

// This code is contributed
// by nitin mittal.
?>
```

Output :

```
Not Same
```

Source: https://www.geeksforgeeks.org/count-of-n-digit-numbers-whose-sum-of-digits-equals-to-given-sum/

Improved By : Smitha Dinesh Semwal, nitin mittal

Source

https://www.geeksforgeeks.org/check-if-two-numbers-are-equal-without-using-arithmetic-and-comparison-operator

Chapter 70

Check if two numbers are equal without using comparison operators

Check if two numbers are equal without using comparison operators - GeeksforGeeks

Following are not allowed to use
1) Comparison Operators
2) String functions

Examples:

```
Input : num1 = 1233, num2 - 1233
Output : Same

Input : num1 = 223, num2 = 233
Output : Not Same
```

Method 1: The idea is to use XOR operator. XOR of two numbers is 0 if the numbers are same, otherwise non-zero.

C++

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

// Finds if a and b are same.
void areSame(int a, int b)
{
   if (a^b)
```

```
        cout << "Not Same";
    else
        cout << "Same";
}

int main()
{
    areSame(10, 20);
}
```

Java

```
 class GFG
{
    // Finds if a and b are same
    static void areSame(int a,int b)
    {
        if( (a ^ b) != 0 )
            System.out.println("Not Same");
        else
            System.out.println("Same");
    }

    public static void main(String args[])
    {
        areSame(10,20);
    }

}
// This code is contributed by Sumit Ghosh
```

Python

```
 # Finds if a and b are same.
def areSame(a, b):
    if (a ^ b):
        print "Not Same"
    else:
        print "Same"

# Driver code
areSame(10, 20)

# This code is submitted by Sachin Bisht
```

C#

```
 // C# program to check if 2
// numbers are same
using System;

class GFG
{
    // Finds if a and b are same
    static void areSame(int a, int b)
    {
        if( (a ^ b) != 0 )
          Console.Write("Not Same");
        else
          Console.Write("Same");
    }

    // Driver code
    public static void Main()
    {

        // Calling Function
        areSame(10, 20);
    }

}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php

// Finds if a and b are same.
function areSame($a, $b)
{
    if ($a ^ $b)
        echo "Not Same";
    else
        echo "Same";
}

// Driver Code
areSame(10, 20);

// This code is contributed by nitin mittal
?>
```

Output:

```
Not Same
```

Method 2: We can subtract the numbers. Same numbers yield 0. If answer is not 0, numbers are not same.

C++

```
 // CPP code to check if 2 numbers are same
#include <bits/stdc++.h>
using namespace std;

// Finds if a and b are same
void areSame(int a, int b)
{
    if (!(a - b))
        cout << "Same";
    else
        cout << "Not Same";
}

// Driver code
int main()
{
     areSame(10, 20);
    return 0;
}
```

Java

```
 // Java code to check if 2 numbers are same
class GFG{

    // Finds if a and b are same
    static void areSame(int a, int b)
    {
        if ((a - b) == 0)
            System.out.println("Same");
        else
            System.out.println("Not Same");
    }

    // Driver code
    public static void main(String args[])
    {
        areSame(10, 20);
```

```
    }
}
//This code is contributed by Sumit Ghosh
```

Python

```
 # Python code to check if 2 numbers are same

# Finds if a and b are same
def areSame(a, b):
    if (not(a - b)):
        print "Same"
    else:
        print "Not Same"
# Driver code
areSame(10, 20)

# This code is submitted by Sachin Bisht
```

C#

```
 // C# code to check if 2
// numbers are same
using System;

class GFG
{

    // Finds if a and b are same
    static void areSame(int a, int b)
    {
        if ((a - b) == 0)
            Console.Write("Same");
        else
            Console.Write("Not Same");
    }

    // Driver code
    public static void Main()
    {

        // Calling Function
        areSame(10, 20);
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP code to check if 2
// numbers are same

// Finds if a and b are same
function areSame($a, $b)
{
    if (!($a - $b))
        echo "Same";
    else
        echo "Not Same";
}

// Driver code
areSame(10, 20);

// This code is contributed by nitin mittal
?>
```

Output:

```
Not Same
```

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/check-two-numbers-equal-without-using-comparison-operators/

Chapter 71

Check in binary array the number represented by a subarray is odd or even

Check in binary array the number represented by a subarray is odd or even - GeeksforGeeks

Given a array such that all its terms is either 0 or 1.You need to tell the number represented by a subarray a[l..r] is odd or even

Examples :

```
Input : arr = {1, 1, 0, 1}
        l = 1, r = 3
Output : odd
        number represented by arr[l...r] is
        101 which 5 in decimal form which is
        odd

Input :  arr = {1, 1, 1, 1}
         l = 0, r = 3
Output : odd
```

The important point to note here is all the odd numbers in binary form have 1 as their rightmost bit and all even numbers have 0 as their rightmost bit.
The reason is simple all other bits other than rightmost bit have even values and sum of even numbers is always even .Now the rightmost bit can have value either 1 or 0 as we know even + odd = odd so when right most bit is 1 the number is odd and when it is 0 the number is even.
So to solve this problem we have to just check if a[r] is 0 or 1 and accordingly print odd or even

C++

```
 // C++ program to find if a subarray
// is even or odd.
#include<bits/stdc++.h>
using namespace std;

// prints if subarray is even or odd
void checkEVENodd (int arr[], int n, int l, int r)
{
    // if arr[r] = 1 print odd
    if (arr[r] == 1)
        cout << "odd" << endl;

    // if arr[r] = 0 print even
    else
        cout << "even" << endl;
}

// driver code
int main()
{
    int arr[] = {1, 1, 0, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    checkEVENodd (arr, n, 1, 3);
    return 0;
}
```

Java

```
 // java program to find if a subarray
// is even or odd.
import java.io.*;

class GFG
{
    // prints if subarray is even or odd
    static void checkEVENodd (int arr[], int n, int l, int r)
    {
        // if arr[r] = 1 print odd
        if (arr[r] == 1)
            System.out.println( "odd") ;

        // if arr[r] = 0 print even
        else
            System.out.println ( "even") ;
    }
```

```
    // driver code
    public static void main (String[] args)
    {
        int arr[] = {1, 1, 0, 1};
        int n = arr.length;
        checkEVENodd (arr, n, 1, 3);


    }
}

// This article is contributed by vt_m.
```

Python3

```
 # Python3 program to find if a
# subarray is even or odd.

# Prints if subarray is even or odd
def checkEVENodd (arr, n, l, r):

    # if arr[r] = 1 print odd
    if (arr[r] == 1):
        print("odd")

    # if arr[r] = 0 print even
    else:
        print("even")

# Driver code
arr = [1, 1, 0, 1]
n = len(arr)
checkEVENodd (arr, n, 1, 3)

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to find if a subarray
// is even or odd.
using System;

class GFG {

    // prints if subarray is even or odd
    static void checkEVENodd (int []arr,
```

```
                    int n, int l, int r)
    {

        // if arr[r] = 1 print odd
        if (arr[r] == 1)
            Console.WriteLine( "odd") ;

        // if arr[r] = 0 print even
        else
            Console.WriteLine( "even") ;
    }

    // driver code
    public static void Main()
    {

        int []arr = {1, 1, 0, 1};
        int n = arr.Length;

        checkEVENodd (arr, n, 1, 3);
    }
}

// This article is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program to find if a subarray
// is even or odd.

// prints if subarray is even or odd
function checkEVENodd ($arr, $n, $l, $r)
{
    // if arr[r] = 1 print odd
    if ($arr[$r] == 1)
        echo "odd", "\n";

    // if arr[r] = 0 print even
    else
        echo "even", "\n";
}

// Driver code
$arr = array(1, 1, 0, 1);
$n = sizeof($arr);
checkEVENodd ($arr, $n, 1, 3);
```

```
// This code is Contributed by Ajit
?>
```

Output :

```
odd
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/check-binary-array-number-represented-subarray-odd-even/

Chapter 72

Check whether K-th bit is set or not

Check whether K-th bit is set or not - GeeksforGeeks

Given a number n, check if k-th bit of n is set or not.

Examples:

```
Input : n = 5, k = 1
Output : SET
5 is represented as 101 in binary
and has its first bit set.

Input : n = 2, k = 3
Output : NOT SET
2 is represented as 10 in binary,
all higher i.e. beyond MSB,
bits are NOT SET.
```

Method 1 (Using Left Shift Operator)
Below are simple steps to find value of k-th bit

```
1) Left shift given number 1 by k-1 to create
   a number that has only set bit as k-th bit.
    temp = 1 << (k-1)
2) If bitwise AND of n and temp is non-zero,
   then result is SET else result is NOT SET.
```

Example:

```
 n = 75 and k = 4
 temp = 1 << (k-1) = 1 << 3 = 8
 Binary Representation of temp = 0..00001000
 Binary Representation of n = 0..01001011
 Since bitwise AND of n and temp is non-zero,
 result is SET.
```

C++

```
 // CPP program to check if k-th bit
// of a given number is set or not
#include <iostream>
using namespace std;

void isKthBitSet(int n, int k)
{
    if (n & (1 << (k - 1)))
        cout << "SET";
    else
        cout << "NOT SET";
}

// Driver code
int main()
{
    int n = 5, k = 1;
    isKthBitSet(n, k);
    return 0;
}
```

Java

```
 // Java program to check if k-th bit
// of a given number is set or not

class Number {
    public static void isKthBitSet(int n,
                                   int k)
    {
        if ((n & (1 << (k - 1))) >= 1)
            System.out.print("SET");
        else
            System.out.print("NOT SET");
    }

    // driver code
```

```
    public static void main(String[] args)
    {
        int n = 5, k = 1;
        isKthBitSet(n, k);
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
 # Python3 code to check if k-th bit
# of a given number is set or not

def isKthBitSet(n, k):
    if n & (1 << (k - 1)):
        print( "SET")
    else:
        print("NOT SET")

# Driver code
n = 5
k = 1
isKthBitSet(n, k)

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# program to check if k-th bit
// of a given number is set or not.
using System;

class GFG {

    public static void isKthBitSet(int n,
                                   int k)
    {
        if ((n & (1 << (k - 1))) >= 1)
            Console.Write("SET");
        else
            Console.Write("NOT SET");
    }

    // Driver code
    public static void Main()
    {
```

```
        int n = 5, k = 1;

        isKthBitSet(n, k);
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to check if
// k-th bit of a given
// number is set or not
function isKthBitSet($n, $k)
{
    if ($n & (1 << ($k - 1)))
        echo "SET";
    else
        echo "NOT SET";
}

// Driver code
$n = 5; $k = 1;
isKthBitSet($n, $k);

// This code is contributed
// by akt_mit
?>
```

Output:

```
SET
```

Method 2 (Using Right Shift Operator)
If we right shift n by k-1, we get last bit as 1 if k-th bit is set else 0.

C++

```
 // CPP program to check if k-th bit
// of a given number is set or not using
// right shift operator.
#include <iostream>
using namespace std;
```

```
void isKthBitSet(int n, int k)
{
    if ((n >> (k - 1)) & 1)
        cout << "SET";
    else
        cout << "NOT SET";
}

// Driver code
int main()
{
    int n = 5, k = 1;
    isKthBitSet(n, k);
    return 0;
}
```

Java

```
 // Java program to check if
// k-th bit of a given number
// is set or not using right
// shift operator.
import java.io.*;

class GFG
{
static void isKthBitSet(int n,
                        int k)
{
    if (((n >> (k - 1)) &
                1) >= 1)
        System.out.println("SET");
    else
        System.out.println("NOT SET");
}

// Driver code
public static void main (String[] args)
{
    int n = 5, k = 1;
    isKthBitSet(n, k);
}
}

// This code is contributed
// by ajit
```

C#

```
 // C# program to check if
// k-th bit of a given number
// is set or not using right
// shift operator
using System;

class GFG
{
static void isKthBitSet(int n,
                        int k)
{
    if (((n >> (k - 1)) &
              1) >= 1)
        Console.WriteLine("SET");
    else
        Console.WriteLine("NOT SET");
}

// Driver code
static public void Main ()
{
    int n = 5, k = 1;
    isKthBitSet(n, k);
}
}

// This code is contributed
// by ajit
```

PHP

```
 <?php
// PHP program to check
// if k-th bit of a given
// number is set or not
// using right shift operator.

function isKthBitSet($n, $k)
{
    if (($n >> ($k - 1)) & 1)
        echo "SET";
    else
        echo "NOT SET";
}
```

```
// Driver code
$n = 5; $k = 1;
isKthBitSet($n, $k);

// This code is contributed
// by akt_mit
?>
```

Output:

```
SET
```

Improved By : nitin mittal, Rajat Rawat 4, jit_t

Source

https://www.geeksforgeeks.org/check-whether-k-th-bit-set-not/

Chapter 73

Check whether a given number is even or odd

Check whether a given number is even or odd - GeeksforGeeks

Given a number, check whether it is even or odd.

A number is called even if it is divisible by 2. All other numbers are odd (not divisible by 2)

First few even numbers : 0, 2, 4, 6, ...
First few odd numbers : 1, 3, 5, 7, ...

Input : n = 12
Output : Even

Input : n = 13
Output : Odd

Examples :

```
input: 2
output: even

input: 5
output: odd
```

One **simple solution** is to find remainder after dividing by 2.

C++

```
 // A simple C++ program to
// check for even or odd
#include <iostream>
using namespace std;

// Returns true if n is
// even, else odd
bool isEven(int n)
{
return (n % 2 == 0);
}

// Driver code
int main()
{
int n = 101;
isEven(n)? cout << "Even" :
           cout << "Odd";

return 0;
}
```

Java

```
 // Java program program to
// check for even or odd

class GFG
{
    // Returns true if n is even, else odd
    public static boolean isEven(int n)
    {
        return (n % 2 == 0);
    }

    // Driver code
    public static void main(String[] args)
```

```
    {
        int n = 101;
        if(isEven(n) == true)
            System.out.print("Even");
        else
            System.out.print("Odd");
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
 # A simple Python3 code
# to check for even or odd

# Returns true if n is even, else odd
def isEven(n):
    return (n % 2 == 0)

# Driver code
n = 101
print("Even" if isEven(n) else "Odd")

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# program program to
// check for even or odd
using System;

class GFG
{
    // Returns true if n is even, else odd
    public static bool isEven(int n)
    {
        return (n % 2 == 0);
    }

    // Driver code
    public static void Main()
    {
        int n = 101;
        if(isEven(n) == true)
            Console.WriteLine("Even");
        else
```

```
            Console.WriteLine("Odd");
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// A simple PHP program to
// check for even or odd

// Returns true if n is
// even, else odd
function isEven($n)
{
    return ($n % 2 == 0);
}

// Driver code
$n = 101;
if(isEven != true)
    echo "Even";
    else
    echo "Odd";

// This code is contributed by Ajit
?>
```

Output :

```
Odd
```

A **better solution** is to use bitwise operators. We need to check whether last bit is 1 or not. If last bit is 1 then number is odd, otherwise always even.

Explanation:

```
 input : 5    // odd
   00000101
 & 00000001
--------------
   00000001
--------------
```

```
input : 8     //even
   00001000
 & 00000001
--------------
   00000000
--------------
```

Below is the implementation of the idea.

C++

```
 // A simple C++ program to
// check for even or odd
#include <iostream>
using namespace std;

// Returns true if n is
// even, else odd
bool isEven(int n)
{

// n & 1 is 1, then
// odd, else even
return (!(n & 1));
}

// Driver code
int main()
{
int n = 101;
isEven(n)? cout << "Even" :
           cout << "Odd";

return 0;
}
```

Java

```
 // Java program program to
// check for even or odd

class GFG
{
    // Returns true if n
    // is even, else odd
    public static boolean isEven(int n)
    {
```

```
        if((n & 1) == 0)
            return true;
        else
            return false;
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 101;
        if(isEven(n) == true)
            System.out.print("Even");
        else
            System.out.print("Odd");
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
 # A Python3 code program
# to check for even or odd

# Returns true if n is even, else odd
def isEven(n):

    # n&1 is 1, then odd, else even
    return (not(n & 1))

# Driver code
n = 101;
print("Even" if isEven(n) else "Odd")

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# program program to
// check for even or odd
using System;

class GFG
{
    // Returns true if n
    // is even, else odd
    public static bool isEven(int n)
```

```
    {
        if((n & 1) == 0)
            return true;
        else
            return false;
    }

    // Driver code
    public static void Main()
    {
        int n = 101;
        if(isEven(n) == true)
            Console.WriteLine("Even");
        else
            Console.WriteLine("Odd");
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// A simple PHP program to
// check for even or odd

// Returns true if n is
// even, else odd
function isEven($n)
{
    return (!($n & 1));
}

// Driver code
$n = 101;
if(isEven($n) == true)
    echo "Even";
else
    echo "Odd";

// This code is contributed by Smitha
?>
```

Output :

```
Odd
```

Improved By : vt_m, jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/check-whether-given-number-even-odd/

Chapter 74

Check whether all the bits are set in the given range

Check whether all the bits are set in the given range - GeeksforGeeks

Given a non-negative number **n** and two values **l** and **r**. The problem is to check whether all the bits are set or not in the range **l** to **r** in the binary representation of **n**.
Constraint: 1 <= l <= r <= number of bits in the binary representation of **n**.

Examples:

```
Input : n = 22, l = 2, r = 3
Output : Yes
(22)10 = (10110)2
The bits in the range 2 to 3 are all set.

Input : n = 47, l = 2, r = 5
Output : No
(47)10 = (101111)2
The bits in the range 2 to 5 are all not set.
```

Approach: Following are the steps:

1. Calculate **num** = ((1 << r) – 1) ^ ((1 << (l-1)) – 1). This will produce a number **num** having **r** number of bits and bits in the range **l** to **r** are the only set bits.
2. Calculate **new_num** = n & num.
3. If num == new_num, return “Yes” (all bits are set in the given range).
4. Else return “No” (all bits are not set in the given range).

C++

```
 // C++ implementation to check whether all the bits
// are set in the given range or not
#include <bits/stdc++.h>

using namespace std;

// function to check whether all the bits
// are set in the given range or not
string allBitsSetInTheGivenRange(unsigned int n,
                                 unsigned int l, unsigned int r)
{
    // calculating a number 'num' having 'r'
    // number of bits and bits in the range l
    // to r are the only set bits
    int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

    // new number which will only have one or more
    // set bits in the range l to r and nowhere else
    int new_num = n & num;

    // if both are equal, then all bits are set
    // in the given range
    if (num == new_num)
        return "Yes";

    // else all bits are not set
    return "No";
}

// Driver program to test above
int main()
{
    unsigned int n = 22;
    unsigned int l = 2, r = 3;
    cout << allBitsSetInTheGivenRange(n, l, r);
    return 0;
}
```

Java

```
 // Java implementation to check whether all
// the bits are set in the given range or not
class GFG {

    // function to check whether all the bits
    // are set in the given range or not
    static String allBitsSetInTheGivenRange(int n,
                                    int l,int r)
```

```
    {

        // calculating a number 'num' having 'r'
        // number of bits and bits in the range
        // l to r are the only set bits
        int num = ((1 << r) - 1) ^ ((1 <<
                                  (l - 1)) - 1);

        // new number which will only have one
        // or more set bits in the range l to r
        // and nowhere else
        int new_num = n & num;

        // if both are equal, then all bits are
        // set in the given range
        if (num == new_num)
            return "Yes";

        // else all bits are not set
        return "No";
    }

    //Driver code
    public static void main (String[] args)
    {
        int n = 22;
        int l = 2, r = 3;

        System.out.print(allBitsSetInTheGivenRange(
                                            n, l, r));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 implementation to check
# whether all the bits are set in
# the given range or not

# Function to check whether all the bits
# are set in the given range or not
def allBitsSetInTheGivenRange(n, l, r):

    # calculating a number 'num' having 'r'
    # number of bits and bits in the range l
    # to r are the only set bits
```

```
    num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1)

    # new number which will only have
    # one or more set bits in the range
    # l to r and nowhere else
    new_num = n & num

    # if both are equal, then all bits
    # are set in the given range
    if (num == new_num):
        return "Yes"

    # else all bits are not set
    return "No"

# Driver code
n, l, r = 22, 2, 3
print(allBitsSetInTheGivenRange(n, l, r))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# implementation to check whether all the bits
// are set in the given range or not
using System;

class GFG
{
    // function to check whether all the bits
    // are set in the given range or not
    static String allBitsSetInTheGivenRange(int n,
                                        int l,int r)
    {
        // calculating a number 'num' having 'r'
        // number of bits and bits in the range l
        // to r are the only set bits
        int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

        // new number which will only have one or more
        // set bits in the range l to r and nowhere else
        int new_num = n & num;

        // if both are equal, then all bits are set
        // in the given range
        if (num == new_num)
            return "Yes";
```

```
        // else all bits are not set
        return "No";
    }

    //Driver code
    public static void Main ()
    {
        int n = 22;
        int l = 2, r = 3;
        Console.Write(allBitsSetInTheGivenRange(n, l, r));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP implementation to check
// whether all the bits are set
// in the given range or not

// function to check whether
// all the bits are set in
// the given range or not
function allBitsSetInTheGivenRange($n, $l, $r)
{

    // Calculating a number
    // 'num' having 'r'
    // number of bits and
    // bits in the range l
    // to r are the only
    // set bits
    $num = ((1 << $r) - 1) ^
           ((1 << ($l - 1)) - 1);

    // new number which will
    // only have one or more
    // set bits in the range
    // l to r and nowhere else
    $new_num = $n & $num;

    // if both are equal,
    // then all bits are set
    // in the given range
    if ($num == $new_num)
        return "Yes";
```

```
    // else all bits
    // are not set
    return "No";
}

    // Driver Code
    $n = 22;
    $l = 2;
    $r = 3;
    echo allBitsSetInTheGivenRange($n, $l, $r);

// This code is contributed by Ajit
?>
```

Output:

```
Yes
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/check-whether-bits-set-given-range/

Chapter 75

Check whether all the bits are unset in the given range

Check whether all the bits are unset in the given range - GeeksforGeeks

Given a non-negative number **n** and two values **l** and **r**. The problem is to check whether all the bits are unset or not in the range **l** to **r** in the binary representation of **n**. The bits are numbered from right to left, i.e., the least significant bit is considered to be at first position.
Constraint: 1 <= l <= r <= number of bits in the binary representation of **n**.

Examples:

```
Input : n = 17, l = 2, r = 4
Output : Yes
(17)10 = (10001)2
The bits in the range 2 to 4 are all unset.

Input : n = 39, l = 4, r = 6
Output : No
(39)10 = (100111)2
The bits in the range 4 to 6 are all not unset.
```

Approach: Following are the steps:

1. Calculate **num** = ((1 << r) – 1) ^ ((1 << (l-1)) – 1). This will produce a number **num** having **r** number of bits and bits in the range **l** to **r** are the only set bits.
2. Calculate **new_num** = n & num.
3. If new_num == 0, return “Yes” (all bits are unset in the given range).
4. Else return “No” (all bits are not unset in the given range).

C++

```
 // C++ implementation to check whether all the bits
// are unset in the given range or not
#include <bits/stdc++.h>

using namespace std;

// function to check whether all the bits
// are unset in the given range or not
bool allBitsSetInTheGivenRange(unsigned int n,
                               unsigned int l, unsigned int r)
{
    // calculating a number 'num' having 'r'
    // number of bits and bits in the range l
    // to r are the only set bits
    int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

    // new number which could only have one or more
    // set bits in the range l to r and nowhere else
    int new_num = n & num;

    // if true, then all bits are unset
    // in the given range
    if (new_num == 0)
        return true;

    // else all bits are not unset
    // in the given range
    return false;
}

// Driver program to test above
int main()
{
    unsigned int n = 17;
    unsigned int l = 2, r = 4;
    if (allBitsSetInTheGivenRange(n, l, r))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // Java implementation to check
// whether all the bits are
// unset in the given range or not
class GFG
```

```
{

// function to check whether
// all the bits are unset in
// the given range or not
static boolean allBitsSetInTheGivenRange(int n,
                                         int l,
                                         int r)
{
    // calculating a number 'num'
    // having 'r' number of bits
    // and bits in the range l
    // to r are the only set bits
    int num = ((1 << r) - 1) ^
              ((1 << (l - 1)) - 1);

    // new number which could only
    // have one or more set bits in
    // the range l to r and nowhere else
    int new_num = n & num;

    // if true, then all bits are
    // unset in the given range
    if (new_num == 0)
        return true;

    // else all bits are not
    // unset in the given range
    return false;
}

// Driver Code
public static void main(String[] args)
{
    int n = 17;
    int l = 2, r = 4;
    if (allBitsSetInTheGivenRange(n, l, r))
        System.out.println("Yes");
    else
        System.out.println("No");
}
}

// This code is contributed
// by Smitha
```

Python3

```
 # Python3 implementation to
# check whether all the bits
# are unset in the given
# range or not

# function to check whether
# all the bits are unset in
# the given range or not
def allBitsSetInTheGivenRange(n, l, r):

    # calculating a number 'num'
    # having 'r' number of bits
    # and bits in the range l
    # to r are the only set bits
    num = (((1 << r) - 1) ^
           ((1 << (l - 1)) - 1))

    # new number which could only
    # have one or more set bits in
    # the range l to r and nowhere else
    new_num = n & num

    # if true, then all bits are
    # unset in the given range
    if (new_num == 0):
        return True

    # else all bits are not
    # unset in the given range
    return false

# Driver Code
n = 17
l = 2
r = 4
if (allBitsSetInTheGivenRange(n, l, r)):
    print("Yes")
else:
    print("No")

# This code is contributed
# by Smitha
```

C#

```
 // C# implementation to check
// whether all the bits are
// unset in the given range or not
```

```
using System;

class GFG
{

// function to check whether
// all the bits are unset in
// the given range or not
static bool allBitsSetInTheGivenRange(int n,
                                      int l,
                                      int r)
{
    // calculating a number 'num'
    // having 'r' number of bits
    // and bits in the range l
    // to r are the only set bits
    int num = ((1 << r) - 1) ^
              ((1 << (l - 1)) - 1);

    // new number which could
    // only have one or more
    // set bits in the range
    // l to r and nowhere else
    int new_num = n & num;

    // if true, then all
    // bits are unset
    // in the given range
    if (new_num == 0)
        return true;

    // else all bits are not
    // unset in the given range
    return false;
}

// Driver Code
public static void Main()
{
    int n = 17;
    int l = 2, r = 4;
    if (allBitsSetInTheGivenRange(n, l, r))
        Console.Write("Yes");
    else
        Console.Write("No");
}
}
```

```
// This code is contributed
// by Smitha
```

PHP

Output:

```
Yes
```

Improved By : Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/check-whether-all-the-bits-are-unset-in-the-given-range/

Chapter 76

Check whether all the bits are unset in the given range or not

Check whether all the bits are unset in the given range or not - GeeksforGeeks

Given a non-negative number **n** and two values **l** and **r**. The problem is to check whether all the bits are unset or not in the range **l** to **r** in the binary representation of **n**.
Constraint: 1 <= l <= r <= number of bits in the binary representation of **n**.

Examples:

```
Input : n = 17, l = 2, r = 4
Output : Yes
(17)10 = (10001)2
The bits in the range 2 to 4 are all unset.

Input : n = 36, l = 3, r = 5
Output : No
(36)10 = (100100)2
The bits in the range 3 to 5 are all not unset.
```

Approach: Following are the steps:

1. Calculate **num** = ((1 << r) – 1) ^ ((1 << (l-1)) – 1). This will produce a number num having **r** number of bits and bits in the range **l** to **r** are the only set bits.
2. Calculate **new_num** = n & num.
3. If new_num == 0, return “Yes” (all bits are unset in the given range).
4. Else return “No” (all bits are not unset in the given range).

C++

```
 // C++ implementation to check whether all
// the bits are unset in the given range
// or not
#include <bits/stdc++.h>
using namespace std;

// function to check whether all the bits
// are unset in the given range or not
string allBitsSetInTheGivenRange(unsigned int n,
                                 unsigned int l,
                                 unsigned int r)
{
    // calculating a number 'num' having 'r'
    // number of bits and bits in the range l
    // to r are the only set bits
    int num = ((1 << r) - 1) ^
              ((1 << (l - 1)) - 1);

    // new number which will only have
    // one or more set bits in the range
    // l to r and nowhere else
    int new_num = n & num;

    // if new num is 0, then all bits
    // are unset in the given range
    return (new_num == 0)
        return "Yes";

    // else all bits are not unset
    return "No";
}

// Driver program to test above
int main()
{
    unsigned int n = 17;
    unsigned int l = 2, r = 4;
    cout << allBitsSetInTheGivenRange(n, l, r);
    return 0;
}
```

Java

```
 // Java implementation to
// check whether all the
// bits are unset in the
// given range or not
import java.io.*;
```

```
class GFG
{

// function to check whether
// all the bits are unset in
// the given range or not
static String allBitsSetInTheGivenRange(int n,
                                        int l,
                                        int r)
{
    // calculating a number 'num'
    // having 'r' number of bits
    // and bits in the range l to
    // r are the only set bits
    int num = ((1 << r) - 1) ^
              ((1 << (l - 1)) - 1);

    // new number which will
    // only have one or more
    // set bits in the range
    // l to r and nowhere else
    int new_num = n & num;

    // if new num is 0, then
    // all bits are unset in
    // the given range
    if(new_num == 0)
        return "Yes";

    // else all bits
    // are not unset
    return "No";
}

// Driver Code
public static void main (String[] args)
{
    int n = 17;
    int l = 2;
    int r = 4;
    System.out.println(
           allBitsSetInTheGivenRange(n, l, r));
}
}

// This code is contributed by akt_mit
```

PHP

```
 <?php
// PHP implementation to check
// whether all the bits are
// unset in the given range
// or not

// function to check whether
// all the bits are unset in
// the given range or not
function allBitsSetInTheGivenRange($n,
                                   $l, $r)
{
    // calculating a number 'num'
    // having 'r' number of bits
    // and bits in the range l
    // to r are the only set bits
    $num = ((1 << $r) - 1) ^
           ((1 << ($l - 1)) - 1);

    // new number which will
    // only have one or more
    // set bits in the range
    // l to r and nowhere else
    $new_num = $n & $num;

    // if new num is 0, then
    // all bits are unset in
    // the given range
    if ($new_num == 0)
        return "Yes";

    // else all bits
    // are not unset
    return "No";
}

// Driver Code
$n = 17;
$l = 2; $r = 4;
echo allBitsSetInTheGivenRange($n,
                               $l, $r);

// This code is contributed
// by ajit
?>
```

Output:

```
Yes
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/check-whether-all-the-bits-are-unset-in-the-given-range-or-not/

Chapter 77

Check whether bits are in alternate pattern in the given range

Check whether bits are in alternate pattern in the given range - GeeksforGeeks

Given a non-negative number **n** and two values **l** and **r**. The problem is to check whether or not **n** has an alternate pattern in its binary representation in the range **l** to **r**. Here alternate pattern means that the set and unset bits occur in alternate order. The bits are numbered from right to left, i.e., the least significant bit is considered to be at first position.

Constraint: 1 <= l <= r <= number of bits in the binary representation of **n**.

Examples:

```
Input : n = 18, l = 1, r = 3
Output : Yes
(18)10 = (10010)2
The bits in the range 1 to 3 in the
binary representation of 18 are in
alternate order.

Input : n = 22, l = 2, r = 4
Output : No
(22)10 = (10110)2
The bits in the range 2 to 4 in the
binary representation of 22 are not in
alternate order.
```

Algorithm:

```
bitsAreInAltPatrnInGivenTRange(n, l, r)

    // Shift bits of given number n by
    // (l-1).
    num = n >> (l - 1)

    // Find first bit of range
    prev = num & 1

    // Traverse through remaining
    // bits. For every bit, check if
    // current bit is same as previous
    // or not.
    num = num >> 1
    for i = 1 to (r - l)
        curr = num & 1

        if curr == prev then
            return false

        prev = curr
        num = num >> 1

    return true;
```

C++

```
 // C++ implementation to check whether bits are in
// alternate pattern in the given range
#include <bits/stdc++.h>

using namespace std;

// function to check whether bits are in
// alternate pattern in the given range
bool bitsAreInAltPatrnInGivenTRange(unsigned int n,
                                    unsigned int l,
                                    unsigned int r)
{
    unsigned int num, prev, curr;

    // right shift n by (l - 1) bits
    num = n >> (l - 1);

    // get the bit at the last position in 'num'
    prev = num & 1;
```

```
    // right shift 'num' by 1
    num = num >> 1;

    // loop until there are bits in the given range
    for (int i = 1; i <= (r - l); i++) {

        // get the bit at the last position in 'num'
        curr = num & 1;

        // if true, then bits are not in alternate
        // pattern
        if (curr == prev)
            return false;

        // update 'prev'
        prev = curr;

        // right shift 'num' by 1
        num = num >> 1;
    }

    // bits are in alternate pattern in the
    // given range
    return true;
}

// Driver program to test above
int main()
{
    unsigned int n = 18;
    unsigned int l = 1, r = 3;

    if (bitsAreInAltPatrnInGivenTRange(n, l, r))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
 // Java implementation to check
// whether bits are in alternate
// pattern in the given range
class GFG
{
```

```
// function to check whether
// bits are in alternate
// pattern in the given range
static boolean bitsAreInAltPatrnInGivenTRange(int n,
                                        int l, int r)
{
    int num, prev, curr;

    // right shift n by (l - 1) bits
    num = n >> (l - 1);

    // get the bit at the
    // last position in 'num'
    prev = num & 1;

    // right shift 'num' by 1
    num = num >> 1;

    // loop until there are
    // bits in the given range
    for (int i = 1; i <= (r - l); i++)
    {

        // get the bit at the
        // last position in 'num'
        curr = num & 1;

        // if true, then bits are
        // not in alternate pattern
        if (curr == prev)
            return false;

        // update 'prev'
        prev = curr;

        // right shift 'num' by 1
        num = num >> 1;
    }

    // bits are in alternate
    // pattern in the given range
    return true;
}

// Driver Code
public static void main(String[] args)
{
    int n = 18;
```

```
    int l = 1, r = 3;

    if (bitsAreInAltPatrnInGivenTRange(n, l, r))
        System.out.println("Yes");
    else
        System.out.println("No");
}
}

// This code is contributed by mits
```

Python3

```
 # Python3 implementation to check
# whether bits are in alternate
# pattern in the given range

# function to check whether
# bits are in alternate
# pattern in the given range
def bitsAreInAltPatrnInGivenTRange(n, l, r):

    # right shift n by (l - 1) bits
    num = n >> (l - 1);

    # get the bit at the
    # last position in 'num'
    prev = num & 1;

    # right shift 'num' by 1
    num = num >> 1;

    # loop until there are
    # bits in the given range
    for i in range(1,(r - l)):

        # get the bit at the
        # last position in 'num'
        curr = num & 1;

        # if true, then bits are
        # not in alternate pattern
        if (curr == prev):
            return False;

        # update 'prev'
        prev = curr;
```

```
        # right shift 'num' by 1
        num = num >> 1;

    # bits are in alternate
    # pattern in the given range
    return True;

# Driver Code
if __name__ == "__main__":
    n = 18;
    l = 1;
    r = 3;

    if(bitsAreInAltPatrnInGivenTRange(n, l, r)):
        print("Yes");
    else:
        print("No");

# This Code is contributed by mits
```

C#

```
// C# implementation to check
// whether bits are in alternate
// pattern in the given range
using System;

class GFG
{

// function to check whether
// bits are in alternate
// pattern in the given range
static bool bitsAreInAltPatrnInGivenTRange(int n,
int l, int r)
{
int num, prev, curr;

// right shift n by (l – 1) bits
num = n >> (l – 1);

// get the bit at the
// last position in ‘num’
prev = num & 1;

// right shift ‘num’ by 1
num = num >> 1;

// loop until there are
// bits in the given range
for (int i = 1; i <= (r - l); i++) { // get the bit at the // last position in 'num' curr = num
```

& 1; // if true, then bits are // not in alternate pattern if (curr == prev) return false; // update 'prev' prev = curr; // right shift 'num' by 1 num = num >> 1;
}

// bits are in alternate
// pattern in the given range
return true;
}

// Driver Code
static void Main()
{
int n = 18;
int l = 1, r = 3;

if (bitsAreInAltPatrnInGivenTRange(n, l, r))
Console.WriteLine("Yes");
else
Console.WriteLine("No");
}
}

// This code is contributed by mits

PHP

```
 <?php
// PHP implementation to check
// whether bits are in alternate
// pattern in the given range

// function to check whether
// bits are in alternate
// pattern in the given range
function bitsAreInAltPatrnInGivenTRange($n, $l, $r)
{

    // right shift n by (l - 1) bits
    $num = $n >> ($l - 1);

    // get the bit at the
    // last position in 'num'
    $prev = $num & 1;

    // right shift 'num' by 1
    $num = $num >> 1;

    // loop until there are
    // bits in the given range
    for ($i = 1; $i <= ($r - $l); $i++)
```

```
    {

        // get the bit at the
        // last position in 'num'
        $curr = $num & 1;

        // if true, then bits are
        // not in alternate pattern
        if ($curr == $prev)
            return false;

        // update 'prev'
        $prev = $curr;

        // right shift 'num' by 1
        $num = $num >> 1;
    }

    // bits are in alternate
    // pattern in the given range
    return true;
}

// Driver Code
$n = 18;
$l = 1;
$r = 3;

if (bitsAreInAltPatrnInGivenTRange($n, $l, $r))
    echo "Yes";
else
    echo "No";

// This Code is contributed by mits
?>
```

Output:

```
Yes
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/check-whether-bits-are-in-alternate-pattern-in-the-given-range/

Chapter 78

Check whether bits are in alternate pattern in the given range | Set-2

Check whether bits are in alternate pattern in the given range | Set-2 - GeeksforGeeks

Given a non-negative number N and two values L and R. The problem is to check whether or not **N** has an alternate pattern in its binary representation in the range **L** to **R**.

Here, alternate pattern means that the set and unset bits occur in alternate order. The bits are numbered from right to left, i.e., the least significant bit is considered to be at first position.

Examples:

```
Input : N = 18, L = 1, R = 3
Output : Yes
(18)10 = (10010)2
The bits in the range 1 to 3 in the
binary representation of 18 are in
alternate order.

Input : N = 22, L = 2, R = 4
Output : No
(22)10 = (10110)2
The bits in the range 2 to 4 in the
binary representation of 22 are not in
alternate order.
```

Simple Approach: It has been discussed in this post which has a worst case time complexity of $O(\log_2 n)$.

Efficient Approach: Following are the steps:

1. Declare two variables **num** and **left_shift**.
2. Check if **rth** bit is set or not in **n**. Refer this post. If set then, assign **num** = n and **left_shift** = r, Else set the **(r+1)th** bit in **n** and assign it to **num**. Refer this post. Also assign **left_shift** = r + 1.
3. Perform **num** = num & ((1 << left_shift) – 1).
4. Perform **num** = num >> (l – 1).
5. Finally check whether bits are in alternate pattern in **num** or not. Refer this post.

The entire idea of the above approach is to create a number **num** in which bits are in same pattern as in the given range of **n** and then check whether bits are in alternate pattern in **num** or not.

```
 // C++ implementation to check whether bits are in
// alternate pattern in the given range
#include <bits/stdc++.h>

using namespace std;

// function to check whether rightmost
// kth bit is set or not in 'n'
bool isKthBitSet(unsigned int n,
                 unsigned int k)
{
    if ((n >> (k - 1)) & 1)
        return true;
    return false;
}

// function to set the rightmost kth bit in 'n'
unsigned int setKthBit(unsigned int n,
                       unsigned int k)
{
    // kth bit of n is being set by this operation
    return ((1 << (k - 1)) | n);
}

// function to check if all the bits are set or not
// in the binary representation of 'n'
bool allBitsAreSet(unsigned int n)
{
    // if true, then all bits are set
    if (((n + 1) & n) == 0)
        return true;

    // else all bits are not set
```

```
    return false;
}

// function to check if a number
// has bits in alternate pattern
bool bitsAreInAltOrder(unsigned int n)
{
    unsigned int num = n ^ (n >> 1);

    // to check if all bits are set
    // in 'num'
    return allBitsAreSet(num);
}

// function to check whether bits are in
// alternate pattern in the given range
bool bitsAreInAltPatrnInGivenRange(unsigned int n,
                                   unsigned int l,
                                   unsigned int r)
{
    unsigned int num, left_shift;

    // preparing a number 'num' and 'left_shift'
    // which can be further used for the check
    // of alternate pattern in the given range
    if (isKthBitSet(n, r)) {
        num = n;
        left_shift = r;
    }
    else {
        num = setKthBit(n, (r + 1));
        left_shift = r + 1;
    }

    // unset all the bits which are left to the
    // rth bit of (r+1)th bit
    num = num & ((1 << left_shift) - 1);

    // right shift 'num' by (l-1) bits
    num = num >> (l - 1);

    return bitsAreInAltOrder(num);
}

// Driver program to test above
int main()
{
    unsigned int n = 18;
```

```
    unsigned int l = 1, r = 3;
    if (bitsAreInAltPatrnInGivenRange(n, l, r))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Output:

```
Yes
```

Time Complexity: O(1).

Source

https://www.geeksforgeeks.org/check-whether-bits-are-in-alternate-pattern-in-the-given-range-set-2/

Chapter 79

Check whether the bit at given position is set or unset

Check whether the bit at given position is set or unset - GeeksforGeeks

Given two positive integers **n** and **k**. The problem is to check whether the bit at position **k** from the right in the binary representation of **n** is set (‘1’) or unset (‘0’).
Constraints: 1 <= k <= number of bits in the binary representation of **n**.

Examples:

```
Input : n = 10, k = 2
Output : Set
(10)10 = (1010)2
The 2nd bit from the right is set.

Input : n = 21, k = 4
Output : Unset
```

Approach: Following are the steps:

1. Calculate **new_num** = (n >> (k – 1)).
2. if (new_num & 1) == 1 then bit is “Set”, else “Unset”.

C++

```
 // C++ implementation to check whether the bit
// at given position is set or unset
#include <bits/stdc++.h>
using namespace std;
```

```
// function to check whether the bit
// at given position is set or unset
bool bitAtGivenPosSetOrUnset(unsigned int n,
                             unsigned int k)
{
    int new_num = n >> (k - 1);

    // if it results to '1' then bit is set,
    // else it results to '0' bit is unset
    return (new_num & 1);
}

// Driver program to test above
int main()
{
    unsigned int n = 10, k = 2;
    if (bitAtGivenPosSetOrUnset(n, k))
        cout << "Set";
    else
        cout << "Unset";
    return 0;
}
```

Java

```
 // Java program to
// check the set bit
// at kth position
import java.io.*;

class GFG {

// function to check whether
// the bit at given position
// is set or unset
static int bitAtGivenPosSetOrUnset
                  ( int n, int k)
{

    // to shift the kth bit
    // at 1st position
    int new_num = n >> (k - 1);

    // Since, last bit is now
    // kth bit, so doing AND with 1
    // will give result.
    return (new_num & 1);
}
```

```
    public static void main (String[] args)
    {
         // K and n must be greater than 0
         int n = 10, k = 2;

    if (bitAtGivenPosSetOrUnset(n, k)==1)
        System.out.println("Set");
    else
        System.out.println("Unset");
    }
}

//This code is contributed by Gitanjali
```

Python3

```
 # python implementation to check
# whether the bit at given
# position is set or unset

import math
#function to check whether the bit
# at given position is set or unset
def bitAtGivenPosSetOrUnset(  n,  k):
     new_num = n >> (k - 1)

     #if it results to '1' then bit is set,
     #else it results to '0' bit is unset
     return (new_num & 1)

# Driver code
n = 10
k = 2
if (bitAtGivenPosSetOrUnset(n, k)):
     print("Set")
else:
    print("Unset")

#This code is contributed by Gitanjali
```

C#

```
 // C# program to check the set bit
// at kth position
using System;

class GFG {
```

```
    // function to check whether
    // the bit at given position
    // is set or unset
    static int bitAtGivenPosSetOrUnset(
                            int n, int k)
    {

        // to shift the kth bit
        // at 1st position
        int new_num = n >> (k - 1);

        // Since, last bit is now
        // kth bit, so doing AND with 1
        // will give result.
        return (new_num & 1);
    }

    // Driver code
    public static void Main ()
    {

        // K and n must be greater
        // than 0
        int n = 10, k = 2;

        if (bitAtGivenPosSetOrUnset(n, k)==1)
            Console.Write("Set");
        else
            Console.Write("Unset");
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP implementation to check whether the bit
// at given position is set or unset

// function to check whether the bit
// at given position is set or unset
function bitAtGivenPosSetOrUnset($n, $k)
{
    $new_num = $n >> ($k - 1);

    // if it results to '1' then bit is set,
```

```
    // else it results to '0' bit is unset
    return ($new_num & 1);
}

    // Driver Code
    $n = 10;
    $k = 2;
    if (bitAtGivenPosSetOrUnset($n, $k))
        echo "Set";
    else
        echo "Unset";

// This code is contributed by Sam007
?>
```

Output:

```
Set
```

Time Complexity: O(1).

Improved By : Sam007

Source

https://www.geeksforgeeks.org/check-whether-bit-given-position-set-unset/

Chapter 80

Check whether the number has only first and last bits set

Check whether the number has only first and last bits set - GeeksforGeeks

Given a positive integer **n**. The problem is to check whether only the first and last bits are set in the binary representation of **n**.

Examples:

```
Input : 9
Output : Yes
(9)10 = (1001)2, only the first and
last bits are set.

Input : 15
Output : No
(15)10 = (1111)2, except first and last
there are other bits also which are set.
```

Approach: Following are the steps:

1. If n == 1, return "Yes".
2. Else check whether n-1 is a power of 2. Refer this post.

C++

```
 // C++ to check whether the number has only
// first and last bits set
#include <bits/stdc++.h>
```

```
using namespace std;

// function to check whether 'n'
// is a power of 2 or not
bool powerOfTwo(unsigned int n)
{
    return (!(n & n-1));
}

// function to check whether the number has only
// first and last bits set
bool onlyFirstAndLastAreSet(unsigned int n)
{
    if (n == 1)
        return true;
    if (n == 2)
        return false;
    return powerOfTwo(n-1);
}

// Driver program to test above
int main()
{
    unsigned int n = 9;

    if (onlyFirstAndLastAreSet(n))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
 // Java program to check whether the
// number has only first and last
// bits set
import java.util.*;

class GFG
{
    // function to check whether 'n'
    // is a power of 2 or not
    static boolean powerOfTwo(int n)
    {
        return ((n & n - 1) == 0);
    }
```

```
    // function to check whether the number has
    // only first and last bits set
    static boolean onlyFirstAndLastAreSet(int n)
    {
        if (n == 1)
            return true;

        return powerOfTwo(n-1);
    }

    // Driver program to test above
    public static void main (String[] args) {

       int n = Integer.parseUnsignedInt("9");

        if (onlyFirstAndLastAreSet(n))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

/* This code is contributed by Mr. Somesh Awasthi */
```

Python3

```
 # Python3 program to check whether number
# has only first and last bits set

# function to check whether 'n'
# is a power of 2 or not
def powerOfTwo (n):
    return (not(n & n-1))


# function to check whether number
# has only first and last bits set
def onlyFirstAndLastAreSet (n):

    if (n == 1):
        return True
    return powerOfTwo (n-1)

# Driver program to test above
n = 9
if (onlyFirstAndLastAreSet (n)):
    print('Yes')
```

```
else:
    print('No')

# This code is contributed by Shariq Raza
```

C#

```
 // C# program to check whether the
// number has only first and last
// bits set
using System;

class GFG {

    // function to check whether 'n'
    // is a power of 2 or not
    static bool powerOfTwo(uint n)
    {
        return ((n & n - 1) == 0);
    }

    // function to check whether the number has
    // only first and last bits set
    static bool onlyFirstAndLastAreSet(uint n)
    {
        if (n == 1)
            return true;

        return powerOfTwo(n - 1);
    }

    // Driver program to test above
    public static void Main()
    {

        uint n = (uint)9;

        if (onlyFirstAndLastAreSet(n))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }

}

// This code is contributed by Sam007
```

Output:

```
Yes
```

Source

https://www.geeksforgeeks.org/check-whether-number-first-last-bits-set/

Chapter 81

Check whether the number has only first and last bits set | Set 2

Check whether the number has only first and last bits set | Set 2 - GeeksforGeeks

Given a positive integer n, check whether only the first and last bits are set in the binary representation of n. Print 'Yes' or 'No'.

Examples:

> Input: 9
> Output: Yes
> (9)10 = (1001)2, only the first and
> last bits are set.
>
> Input: 15
> Output: No
> (15)10 = (1111)2, except first and last
> there are other bits also which are set.

We have already discussed a solution here.

In this post, a simpler solution is discussed.

C++

```
 // C++ to check whether the number has only
// first and last bits set
#include <bits/stdc++.h>
using namespace std;

// function to check whether the number has only
// first and last bits set
bool onlyFirstAndLastAreSet(unsigned int n)
```

```
{
    if (n == 1)
        return true;
    if (n == 2)
        return false;
    return (((n - 1) & (n - 2)) == 0);
}

// Driver program to test above
int main()
{
    unsigned int n = 9;
    if (onlyFirstAndLastAreSet(n))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
 // Java to check whether
// the number has only
// first and last bits set

class GFG
{
// function to check whether
// the number has only
// first and last bits set
static boolean onlyFirstAndLastAreSet(int n)
{
    if (n == 1)
        return true;
    if (n == 2)
        return false;
    return (((n - 1) &
             (n - 2)) == 0);
}

// Driver Code
public static void main(String[] args)
{
    int n = 9;
    if (onlyFirstAndLastAreSet(n))
        System.out.println("Yes");
    else
```

```
        System.out.println("No");
}
}

// This code is contributed
// by Smitha
```

Python3

```
 # Python 3 to check whether
# the number has only
# first and last bits set

# function to check whether
# the number has only
# first and last bits set
def onlyFirstAndLastAreSet(n):

    if (n == 1):
        return True
    if (n == 2):
        return False

    return (((n - 1) &
             (n - 2)) == 0)

# Driver Code
n = 9
if (onlyFirstAndLastAreSet(n)):
    print("Yes")
else:
    print("No")

# This code is contributed
# by Smitha
```

C#

```
 // C# to check whether
// the number has only
// first and last bits set
using System;

class GFG
{
// function to check whether
// the number has only
```

```
// first and last bits set
static bool onlyFirstAndLastAreSet(int n)
{
    if (n == 1)
        return true;
    if (n == 2)
        return false;
    return (((n - 1) &
             (n - 2)) == 0);
}

// Driver Code
public static void Main()
{
    int n = 9;
    if (onlyFirstAndLastAreSet(n))
        Console.Write("Yes");
    else
        Console.Write("No");
}
}

// This code is contributed
// by Smitha
```

PHP

```
 <?php
// PHP to check whether the
// number has only first and
// last bits set

// function to check whether
// the number has only first
// and last bits set
function onlyFirstAndLastAreSet($n)
{
    if ($n == 1)
        return true;
    if ($n == 2)
        return false;
    return (($n - 1) &
             ($n - 2)) == 0);
}

// Driver Code
$n = 9;
if (onlyFirstAndLastAreSet($n))
```

```
    echo "Yes";
else
    echo "No";

// This code is contributed
// by Smitha
?>
```

Output:

```
Yes
```

Improved By : Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/check-whether-the-number-has-only-first-and-last-bits-set-set-2/

Chapter 82

Check whether the two numbers differ at one bit position only

Check whether the two numbers differ at one bit position only - GeeksforGeeks

Given two non-negative integers **a** and **b**. The problem is to check whether the two numbers differ at one bit position only or not.

Examples:

```
Input : a = 13, b = 9
Output : Yes
(13)10 = (1101)2
(9)10 = (1001)2
Both the numbers differ at one bit position only, i.e,
differ at the 3rd bit from the right.

Input : a = 15, b = 8
Output : No
```

Approach: Following are the steps:

1. Calculate **num** = a ^ b.
2. Check whether **num** is a power of 2 or not. Refer this post.

CPP

```
 // C++ implementation to check whether the two
// numbers differ at one bit position only
#include <bits/stdc++.h>
using namespace std;
```

```
// function to check if x is power of 2
bool isPowerOfTwo(unsigned int x)
{
    // First x in the below expression is
    // for the case when x is 0
    return x && (!(x & (x - 1)));
}

// function to check whether the two numbers
// differ at one bit position only
bool differAtOneBitPos(unsigned int a,
                       unsigned int b)
{
    return isPowerOfTwo(a ^ b);
}

// Driver program to test above
int main()
{
    unsigned int a = 13, b = 9;
    if (differAtOneBitPos(a, b))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Python3

```
 # Python3 implementation to check whether the two
# numbers differ at one bit position only

# function to check if x is power of 2
def isPowerOfTwo( x ):

    # First x in the below expression is
    # for the case when x is 0
    return x and (not(x & (x - 1)))

# function to check whether the two numbers
# differ at one bit position only
def differAtOneBitPos( a , b ):
    return isPowerOfTwo(a ^ b)

# Driver code to test above
a = 13
b = 9
```

```
if (differAtOneBitPos(a, b)):
    print("Yes")
else:
    print( "No")

# This code is contributed by "Sharad_Bhardwaj".
```

PHP

```
 <?php
// PHP implementation to check
// whether the two numbers differ
// at one bit position only

    // function to check if x is power of 2
    function isPowerOfTwo($x)
    {
        $y = 0;

        // First x in the below expression is
        // for the case when x is 0
        if($x && (!($x & ($x - 1))))
        $y = 1;
        return $y;
    }

    // function to check whether
    // the two numbers differ at
    // one bit position only
    function differAtOneBitPos($a,$b)
    {
        return isPowerOfTwo($a ^ $b);
    }

        // Driver Code
        $a = 13;
        $b = 9;
        if (differAtOneBitPos($a, $b))
            echo "Yes";
        else
            echo "No";

// This code is contributed by Sam007
?>
```

Output:

```
Yes
```

Time Complexity: O(1).

Improved By : Sam007

Source

https://www.geeksforgeeks.org/check-whether-two-numbers-differ-one-bit-position/

Chapter 83

Closest (or Next) smaller and greater numbers with same number of set bits

Closest (or Next) smaller and greater numbers with same number of set bits - GeeksforGeeks

Given a positive integer n, print the next smallest and the previous largest number that have the same number of 1 bits in their binary representation.

Examples :

```
Input : n = 5
Output : Closest Greater = 6
         Closest Smaller = 3
Note that 5, 6 and 3 have same number of
set bits.

Input : n = 11
Output : Closest Greater = 13
         Closest Smaller = 7
```

The Brute Force Approach
An easy approach is simply brute force: count the number of 1s in n, and then increment (or decrement) until we find a number with the same number of 1s.

Optimal Approaches
Let's start with the code for getNext, and then move on to getPrev.

Bit Manipulation Approach for Get Next Number
If we think about what the next number should be, we can observe the following. Given the number 13948, the binary representation looks like:

```
1   1   0   1   1  0  0  1  1  1  1  1  0  0
13  12  11  10  9  8  7  6  5  4  3  2  1  0
```

We want to make this number bigger (but not too big). We also need to keep the same number of ones.
Observation: Given a number n and two bit locations i and j, suppose we flip bit i from a 1 to a 0, and bit j from a 0 to a 1. If i > j, then n will have decreased. If i < j, then n will have increased.

We know the following:

- If we flip a zero to a one, we must flip a one to a zero.
- The number (After two flips) will be bigger if and only if the zero-to-one bit was to the left of the one to zero bit.
- We want to make the number bigger, but not unnecessarily bigger. Therefore, we need to flip the rightmost zero which has ones on the right of it.

To put this in a different way, we are flipping the rightmost non-trailing zero. That is, using the above example, the trailing zeros are in the 0th and 1st spot. The rightmost non-trailing zero is at bit 7. Let's call this position p.

```
p ==> Position of rightmost non-trailing 0.
```

Step 1: Flip rightmost non-trailing zero

```
1    1   0   1  1  0  1  1  1  1  1  1  0  0
13  12  11  10  9  8  7  6  5  4  3  2  1  0
```

With this change, we have increased the number of 1s of n. We can shrink the number by rearranging all the bits to the right of bit p such that the 0s are on the left and the 1s are on the right. As we do this, we want to replace one of the 1s with a 0.

A relatively easy way of doing this is to count how many ones are to the right of p, clear all the bits from 0 until p, and then add back in c1-1 ones. Let c1 be the number of ones to the right of p and c0 be the number of zeros to the right of p.

Let's walk through this with an example.

```
c1 ==> Number of ones to the right of p
c0 ==> Number of zeros to the right of p.
p = c0 + c1
```

Step 2: Clear bits to the right of p. From before, c0 = 2. c1 = 5. p = 7.

```
1    1   0   1  1  0  1  0  0  0  0  0  0  0
13  12  11  10  9  8  7  6  5  4  3  2  1  0
```

To clear these bits, we need to create a mask that is a sequence of ones, followed by p zeros. We can do this as follows:

```
// all zeros except for a 1 at position p.
a = 1 << p;

// all zeros, followed by p ones.
b = a - 1;

// all ones, followed by p zeros.
mask = ~b;

// clears rightmost p bits.
n = n & mask;

Or, more concisely, we do:
n &= ~((1 << p) - 1).
```

Step 3: Add one c1 – 1 ones.

```
1   1   0   1   1  0  1  0  0  0  1  1  1  1
13  12  11  10  9  8  7  6  5  4  3  2  1  0
```

To insert c1 – 1 ones on the right, we do the following:

```
// 0s with a 1 at position c1- 1
a = 1 << (c1 - 1);

// 0s with 1s at positions 0 through c1-1
b = a - 1;

// inserts 1s at positions 0 through c1-1
n = n | b;

Or, more concisely:
n | = (1 << (c1 - 1)) - 1;
```

We have now arrived at the smallest number bigger than n with the same number of ones. The Implementation of code for getNext is below

C++

```
 // C++ implementation of  getNext with
// same number of bits 1's is below
#include <bits/stdc++.h>
using namespace std;

// Main Function to find next smallest
// number bigger than n
int getNext(int n)
{
    /* Compute c0 and c1 */
    int c = n;
    int c0 = 0;
    int c1 = 0;

    while (((c & 1) == 0) && (c != 0))
    {
        c0 ++;
        c >>= 1;
    }
    while ((c & 1)==1)
    {
        c1++;
        c >>= 1;
    }

    // If there is no bigger number with the
    // same no. of 1's
    if (c0 +c1 == 31 || c0 +c1== 0)
        return -1;

    // position of rightmost non-trailing zero
    int p = c0 + c1;

    // Flip rightmost non-trailing zero
    n |= (1 << p);

    // Clear all bits to the right of p
    n &= ~((1 << p) - 1);

    // Insert (c1-1) ones on the right.
    n |= (1 << (c1 - 1)) - 1;

    return n;
}

// Driver Code
int main()
{
```

```
    int n = 5;   // input 1
    cout << getNext(n) << endl;

    n = 8;     // input 2
    cout << getNext(n);
    return 0;
}
```

Java

```
 // Java implementation of
// getNext with same number
// of bits 1's is below
import java.io.*;

class GFG
{

    // Main Function to find next
    // smallest number bigger than n
    static int getNext(int n)
    {

        /* Compute c0 and c1 */
        int c = n;
        int c0 = 0;
        int c1 = 0;

        while (((c & 1) == 0) &&
                (c != 0))
        {
            c0++;
            c >>= 1;
        }
        while ((c & 1) == 1)
        {
            c1++;
            c >>= 1;
        }

        // If there is no bigger number
        // with the same no. of 1's
        if (c0 + c1 == 31 ||
            c0 + c1 == 0)
            return -1;

        // position of rightmost
        // non-trailing zero
```

```
        int p = c0 + c1;

        // Flip rightmost
        // non-trailing zero
        n |= (1 << p);

        // Clear all bits
        // to the right of p
        n &= ~((1 << p) - 1);

        // Insert (c1-1) ones
        // on the right.
        n |= (1 << (c1 - 1)) - 1;

        return n;
    }


    // Driver Code
    public static void main (String[] args)
    {

        int n = 5; // input 1
        System.out.println(getNext(n));

        n = 8; // input 2
        System.out.println(getNext(n));
    }
}

// This code is contributed by aj_36
```

C#

```
 // C# implementation of getNext with
// same number of bits 1's is below
using System;

class GFG {

    // Main Function to find next
    // smallest number bigger than n
    static int getNext(int n)
    {

        /* Compute c0 and c1 */
        int c = n;
        int c0 = 0;
```

```
        int c1 = 0;

        while (((c & 1) == 0) && (c != 0))
        {
            c0++;
            c >>= 1;
        }
        while ((c & 1) == 1)
        {
            c1++;
            c >>= 1;
        }

        // If there is no bigger number
        // with the same no. of 1's
        if (c0 + c1 == 31 || c0 + c1== 0)
            return -1;

        // position of rightmost
        // non-trailing zero
        int p = c0 + c1;

        // Flip rightmost non-trailing
        // zero
        n |= (1 << p);

        // Clear all bits to the right
        // of p
        n &= ~((1 << p) - 1);

        // Insert (c1-1) ones on the
        // right.
        n |= (1 << (c1 - 1)) - 1;

        return n;
    }

    // Driver Code
    static void Main()
    {
        int n = 5; // input 1
        Console.WriteLine(getNext(n));

        n = 8; // input 2
        Console.Write(getNext(n));

    }
}
```

```
// This code is contributed by Anuj_67
```

PHP

```
 <?php
// PHP implementation of getNext with
// same number of bits 1's is below

// Function to find next smallest
// number bigger than n
function getNext($n)
{

    // Compute c0 and c1
    $c = $n;
    $c0 = 0;
    $c1 = 0;

    while ((($c & 1) == 0) &&
                   ($c != 0))
    {
        $c0 ++;
        $c >>= 1;
    }
    while (($c & 1) == 1)
    {
        $c1++;
        $c >>= 1;
    }

    // If there is no bigger
    // number with the
    // same no. of 1's
    if ($c0 + $c1 == 31 ||
        $c0 + $c1== 0)
        return -1;

    // position of rightmost
    // non-trailing zero
    $p = $c0 + $c1;

    // Flip rightmost non -
    // trailing zero
    $n |= (1 << $p);

    // Clear all bits to
    // the right of p
```

```
    $n &= ~((1 << $p) - 1);

    // Insert (c1-1) ones
    // on the right.
    $n |= (1 << ($c1 - 1)) - 1;

    return $n;
}

    // Driver Code
    // input 1
    $n = 5;
    echo getNext($n),"\n";

    // input 2
    $n = 8;
    echo getNext($n);

// This code is contributed by ajit
?>
```

Output:

```
6
16
```

Optimal Bit Manipulation Approach for Get Previous Number
To implement getPrev, we follow a very similar approach.

- Compute c0 and c1. Note that **c1** is the number of trailing ones, and **c0** is the size of the block of zeros immediately to the left of the trailing ones.
- Flip the rightmost non-trailing one to a zero. This will be at position p = c1 + c0.
- Clear all bits to the right of bit p.
- Insert c1 + 1 ones immediately to the right of position p.

Note that Step 2 sets bit p to a zero and Step 3 sets bits 0 through p-1 to a zero. We can merge these steps.

Let's walk through this with an example.

```
c1 ==> number of trailing ones
c0 ==> size of the block of zeros immediately
       to the left of the trailing ones.
p = c1 + c0
```

Step 1: Initial Number: p = 7. c1 = 2. c0 = 5.

```
1   0   0   1   1  1  1  0  0  0  0  0  1  1
13  12  11  10  9  8  7  6  5  4  3  2  1  0
```

Steps 2 & 3: Clear bits 0 through p.

```
1   0   0   1   1  1  0  0  0  0  0  0  0  0
13  12  11  10  9  8  7  6  5  4  3  2  1  0
```

We can do this as follows:

```
// Sequence of 1s
int a = ~0;

// Sequence of 1s followed by p + 1 zeros.
int b = a << (p + 1);

// Clears bits 0 through p.
n & = b;
```

Steps 4: Insert c1 + 1 ones immediately to the right of position p.

```
1   0   0   1   1  1  0  1  1  1  0  0  0  0
13  12  11  10  9  8  7  6  5  4  3  2  1  0
```

Note that since p =c1 + c0, then (c1 + 1) ones will be followed by (c0 – 1)zeros.

We can do this as follows:

```
// 0s with 1 at position (c1 + 1)
int a = 1 << (c1 + 1);

// 0s followed by c1 + 1 ones
int b = a - 1;

// c1 + 1 ones followed by c0 - 1 zeros.
int c = b << (c0 - 1);
n |= c;
```

The code to implement this is below.

C++

```
 // C++ Implementation of  getPrev in
// Same number of bits 1's is below
#include <bits/stdc++.h>
using namespace std;

// Main Function to find next Bigger number
// Smaller than n
int getPrev(int n)
{
    /* Compute c0 and c1  and store N*/
    int temp = n;
    int c0 = 0;
    int c1= 0;

    while ((temp & 1) == 1)
    {
        c1++;
        temp = temp >> 1;
    }

    if (temp == 0)
        return -1;

    while (((temp & 1) == 0) && (temp!= 0))
    {
        c0++;
        temp  = temp >> 1;
    }

    // position of rightmost non-trailing one.
    int p = c0 + c1;

    // clears from bit p onwards
    n = n & ((~0) << (p + 1));

    // Sequence of (c1+1) ones
    int mask = (1 << (c1 + 1)) - 1;

    n = n | mask << (c0 - 1);

    return n;
}

// Driver Code
```

```
int main()
{
    int n = 6;   // input 1
    cout << getPrev(n);

    n = 16;     // input 2
    cout << endl;
    cout << getPrev(n);

    return 0;
}
```

Java

```
 // Java Implementation of
// getPrev in Same number
// of bits 1's is below
import java.io.*;

class GFG
{

// Main Function to find
// next Bigger number
// Smaller than n
static int getPrev(int n)
{
    // Compute c0 and
    // c1 and store N
    int temp = n;
    int c0 = 0;
    int c1= 0;

    while((temp & 1) == 1)
    {
        c1++;
        temp = temp >> 1;
    }

    if(temp == 0)
        return -1;

    while(((temp & 1) == 0) &&
           (temp!= 0))
    {
        c0++;
        temp = temp >> 1;
    }
```

```
    // position of rightmost
    // non-trailing one.
    int p = c0 + c1;

    // clears from bit p onwards
    n = n & ((~0) << (p + 1));

    // Sequence of (c1+1) ones
    int mask = (1 << (c1 + 1)) - 1;

    n = n | mask << (c0 - 1);

    return n;
}

// Driver Code
public static void main(String[] args)
{
    int n = 6; // input 1
    System.out.println(getPrev(n));

    n = 16; // input 2
    System.out.println(getPrev(n));
}
}

// This code is contributed by aj_36
```

C#

```
 // C# Implementation of
// getPrev in Same number
// of bits 1's is below
using System;

class GFG
{

// Main Function to find
// next Bigger number
// Smaller than n
static int getPrev(int n)
{
    // Compute c0 and
    // c1 and store N
    int temp = n;
    int c0 = 0;
```

```
    int c1 = 0;

    while((temp & 1) == 1)
    {
        c1++;
        temp = temp >> 1;
    }

    if(temp == 0)
        return -1;

    while(((temp & 1) == 0) &&
           (temp != 0))
    {
        c0++;
        temp = temp >> 1;
    }

    // position of rightmost
    // non-trailing one.
    int p = c0 + c1;

    // clears from
    // bit p onwards
    n = n & ((~0) << (p + 1));

    // Sequence of
    // (c1+1) ones
    int mask = (1 << (c1 + 1)) - 1;

    n = n | mask << (c0 - 1);

    return n;
}

// Driver Code
static public void Main ()
{
    int n = 6; // input 1
    Console.WriteLine(getPrev(n));

    n = 16; // input 2
    Console.WriteLine(getPrev(n));
}
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP Implementation of getPrev in
// Same number of bits 1's is below

// Main Function to find next Bigger
// number Smaller than n

function getPrev($n)
{
    // Compute c0 and
    // c1 and store N
    $temp = $n;
    $c0 = 0;
    $c1= 0;

    while (($temp & 1) == 1)
    {
        $c1++;
        $temp = $temp >> 1;
    }

    if ($temp == 0)
        return -1;

    while ((($temp & 1) == 0) &&
            ($temp!= 0))
    {
        $c0++;
        $temp = $temp >> 1;
    }

    // position of rightmost
    // non-trailing one.
    $p = $c0 + $c1;

    // clears from bit p onwards
    $n = $n & ((~0) << ($p + 1));

    // Sequence of (c1 + 1) ones
    $mask = (1 << ($c1 + 1)) - 1;

    $n = $n | $mask << ($c0 - 1);

    return $n;
}
```

```
// Driver Code

// input 1
$n = 6;
echo getPrev($n);

// input 2
$n = 16;
echo " \n" ;
echo getPrev($n);

// This code is contributed by Ajit
?>
```

Output :

```
5
8
```

Arithmetic Approach to Get Next Number

If c0 is the number of trailing zeros, c1 is the size of the one block immediately following, and p = c0 + c1, we can form our solution from earlier as follows:

1. Set the p-th bit to 1.
2. Set all bits following p to 0.
3. Set bits 0 through c1 – 2 to 1. This will be c1 – 1 total bits.

A quick way to perform steps 1 and 2 is to set the trailing zeros to 1 (giving us p trailing ones), and then add 1. Adding one will flip all trailing ones, so we wind up with a 1 at bit p followed by p zeros. We can perform this arithmetically.

// Sets trailing 0s to 1, giving us p trailing 1s.
n += $2^{c0} - 1$;

// Flips first p ls to 0s, and puts a 1 at bit p.
n += 1;

Now, to perform Step 3 arithmetically, we just do:

// Sets trailing c1 – 1 zeros to ones.
n += $2^{c1 - 1} - 1$;

This math reduces to:
next = n + $(2^{c0} - 1)$ + 1 + $(2^{c1 - 1} - 1)$
= n + 2^{c0} + $2^{c1 - 1}$ – 1

The best part is that, using a little bit manipulation, it's simple to code.

C++

```
 // C++ Implementation of getNext with
// Same number of bits 1's is below
#include <bits/stdc++.h>
using namespace std;

// Main Function to find next smallest number
// bigger than n
int getNext(int n)
{
    /* Compute c0 and c1 */
    int c = n;
    int c0 = 0;
    int c1 = 0;

    while (((c & 1) == 0) && (c != 0))
    {
        c0 ++;
        c >>= 1;
    }
    while ((c & 1)==1)
    {
        c1++;
        c >>= 1;
    }

    // If there is no bigger number with the
    // same no. of 1's
    if (c0 +c1 == 31 || c0 +c1== 0)
        return -1;

    return n + (1 << c0) + (1 << (c1 - 1)) - 1;
}

// Driver Code
int main()
{
    int n = 5; // input 1
    cout << getNext(n);

    n = 8;     // input 2
    cout << endl;
    cout << getNext(n);
    return 0;
}
```

Java

```
 // Java Implementation of getNext with
```

```
// Same number of bits 1's is below
import java.io.*;

class GFG
{

// Function to find next smallest
// number bigger than n
static int getNext(int n)
{
    /* Compute c0
    and c1 */
    int c = n;
    int c0 = 0;
    int c1 = 0;

    while (((c & 1) == 0) && (c != 0))
    {
        c0 ++;
        c >>= 1;
    }
    while ((c & 1) == 1)
    {
        c1++;
        c >>= 1;
    }

    // If there is no bigger number
    // with the same no. of 1's
    if (c0 + c1 == 31 || c0 + c1 == 0)
        return -1;

    return n + (1 << c0) +
               (1 << (c1 - 1)) - 1;
}

// Driver Code
public static void main (String[] args)
{
int n = 5; // input 1
System.out.println(getNext(n));

n = 8; // input 2
System.out.println(getNext(n));
}
}

// This code is contributed by ajit
```

C#

```
 // C# Implementation of getNext
// with Same number of bits
// 1's is below
using System;

class GFG
{

// Function to find next smallest
// number bigger than n
static int getNext(int n)
{
    /* Compute c0
    and c1 */
    int c = n;
    int c0 = 0;
    int c1 = 0;

    while (((c & 1) == 0) &&
            (c != 0))
    {
        c0 ++;
        c >>= 1;
    }
    while ((c & 1) == 1)
    {
        c1++;
        c >>= 1;
    }

    // If there is no bigger
    // number with the same
    // no. of 1's
    if (c0 + c1 == 31 ||
        c0 + c1 == 0)
        return -1;

    return n + (1 << c0) +
               (1 << (c1 - 1)) - 1;
}

// Driver Code
static public void Main ()
{
    int n = 5; // input 1
    Console.WriteLine(getNext(n));
```

```
    n = 8; // input 2
    Console.WriteLine(getNext(n));
    }
}

// This code is contributed by m_kit
```

PHP

```
 <?php
// PHP Implementation of
// getNext with Same number
// of bits 1's is below

// Main Function to find
// next smallest number
// bigger than n
function getNext($n)
{
    /* Compute c0 and c1 */
    $c = $n;
    $c0 = 0;
    $c1 = 0;

    while ((($c & 1) == 0) &&
            ($c != 0))
    {
        $c0 ++;
        $c >>= 1;
    }
    while (($c & 1) == 1)
    {
        $c1++;
        $c >>= 1;
    }

    // If there is no bigger
    // number with the
    // same no. of 1's
    if ($c0 + $c1 == 31 ||
        $c0 + $c1 == 0)
        return -1;

    return $n + (1 << $c0) +
                (1 << ($c1 - 1)) - 1;
}
```

```
// Driver Code
$n = 5; // input 1
echo getNext($n);

$n = 8; // input 2
echo "\n";
echo getNext($n);

// This code is contributed by ajit
?>
```

Output :

```
6
16
```

Arithmetic Approach to Get Previous Number
If c1 is the number of trailing ones, c0 is the size of the zero block immediately following, and p =c0 + c1, we can word the initial getPrev solution as follows:

1. Set the pth bit to 0
2. Set all bits following p to 1
3. Set bits 0 through c0 – 1 to 0.

We can implement this arithmetically as follows. For clarity in the example, we assume n = 10000011. This makes c1 = 2 and c0 = 5.

```
// Removes trailing 1s. n is now 10000000.
n -= 2c1 - 1;

// Flips trailing 0s. n is now 01111111.
n -= 1;

// Flips last (c0-1) 0s. n is now 01110000.
n -= 2c0 - 1 - 1;

This reduces mathematically to:
next = n - (2c1 - 1) - 1 - ( 2c0-1 - 1) .
     = n - 2c1 - 2c0-1 + 1;
```

Again, this is very easy to implement.

C++

```
 // C++ Implementation of Arithmetic Approach to
// getPrev with Smae number of bits 1's is below
#include <bits/stdc++.h>
using namespace std;

// Main Function to find next Bigger number
// Smaller than n
int getPrev(int n)
{
    /* Compute c0 and c1  and store N*/
    int temp = n;
    int c0 = 0;
    int c1 = 0;

    while ((temp & 1) == 1)
    {
        c1++;
        temp = temp >> 1;
    }

    if (temp == 0)
        return -1;

    while (((temp & 1) == 0) && (temp!= 0))
    {
        c0++;
        temp  = temp >> 1;
    }

    return n - (1 << c1) - (1 << (c0 - 1)) + 1;
}

// Driver Code
int main()
{
    int n = 6;   // input 1
    cout << getPrev(n);

    n = 16;     // input 2
    cout << endl;
    cout << getPrev(n);

    return 0;
}
```

Java

```
 // Java Implementation of Arithmetic
```

```
// Approach to getPrev with Same
// number of bits 1's is below
import java.io.*;

class GFG
{

// Main Function to find next
// Bigger number Smaller than n
static int getPrev(int n)
{
    /* Compute c0 and
    c1 and store N*/
    int temp = n;
    int c0 = 0;
    int c1 = 0;

    while ((temp & 1) == 1)
    {
        c1++;
        temp = temp >> 1;
    }

    if (temp == 0)
        return -1;

    while (((temp & 1) == 0) &&
            (temp!= 0))
    {
        c0++;
        temp = temp >> 1;
    }

    return n - (1 << c1) -
               (1 << (c0 - 1)) + 1;
}

// Driver Code
public static void main (String[] args)
{

    int n = 6; // input 1
    System.out.println (getPrev(n));

    n = 16; // input 2
    System.out.println(getPrev(n));

}
```

```
}

// This code is contributed by akt_mit
```

C#

```
 // C# Implementation of Arithmetic
// Approach to getPrev with Same
// number of bits 1's is below
using System;

class GFG
{

// Main Function to find next
// Bigger number Smaller than n
static int getPrev(int n)
{
    /* Compute c0 and
    c1 and store N*/
    int temp = n;
    int c0 = 0;
    int c1 = 0;

    while ((temp & 1) == 1)
    {
        c1++;
        temp = temp >> 1;
    }

    if (temp == 0)
        return -1;

    while (((temp & 1) == 0) &&
            (temp!= 0))
    {
        c0++;
        temp = temp >> 1;
    }

    return n - (1 << c1) -
               (1 << (c0 - 1)) + 1;
}

// Driver Code
static public void Main ()
{
    int n = 6; // input 1
```

```
    Console.WriteLine(getPrev(n));

    n = 16; // input 2
    Console.WriteLine(getPrev(n));
}
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP program to count of
// steps until one of the
// two numbers become 0.

// Returns count of steps
// before one of the numbers
// become 0 after repeated
// subtractions.
function countSteps($x, $y)
{
    // If y divides x, then
    // simply return x/y.
    if ($x % $y == 0)
        return floor(((int)$x / $y));

    // Else recur. Note that this
    // function works even if x is
    // smaller than y because in that
    // case first recursive call
    // exchanges roles of x and y.
    return floor(((int)$x / $y) +
                  countSteps($y, $x % $y));
}

// Driver code
$x = 100;
$y = 19;
echo countSteps($x, $y);

// This code is contributed by aj_36
?>
```

Output :

```
5
8
```

Improved By : jit_t, vt_m

Source

https://www.geeksforgeeks.org/closest-next-smaller-greater-numbers-number-set-bits/

Chapter 84

Compare two integers without using any Comparison operator

Compare two integers without using any Comparison operator - GeeksforGeeks

Given two integers A & B. Task is to check if A and B are same or not without using comparison operators.

Examples:

```
Input : A = 5 , B = 6
Output : 0

Input : A = 5 , B = 5
Output : 1

Note : 1 = "YES" and 0 = "NO"
```

idea is pretty simple we do Xor of both element (A , B) . if Xor is zero then two number are equal else not .
Below is the implementation of above idea :

C++

```
 // C++ program to compare two integers witout
// any comparison operator.
#include<bits/stdc++.h>
using namespace std;

// function return true if A ^ B > 0  else false
bool EqualNumber(int A, int B)
```

```
{
   return ( A ^ B ) ;
}

// Driver program
int main()
{
  int A = 5 , B = 6;
  cout << !EqualNumber(A, B) << endl;
  return 0;
}
```

Python3

```
 # Python3 program to compare two integers
# without any comparison operator.

# Function return true if
# A ^ B > 0 else false
def EqualNumber(A, B):

    return ( A ^ B )

# Driver Code
A = 5; B = 6
print(int(not(EqualNumber(A, B))))

# This code is contributed by Smitha Dinesh Semwal.
```

Output:

```
0
```

Reference :http://stackoverflow.com/questions/476800/comparing-two-integers-without-any-comparison

Source

https://www.geeksforgeeks.org/compare-two-integers-without-using-comparison-operator/

Chapter 85

Compute modulus division by a power-of-2-number

Compute modulus division by a power-of-2-number - GeeksforGeeks

Compute n modulo d without division(/) and modulo(%) operators, where d is a power of 2 number.

Let *i*th bit from right is set in d. For getting n modulus d, we just need to return 0 to *i*-1 (from right) bits of n as they are and other bits as 0.

For example if n = 6 (00..110) and d = 4(00..100). Last set bit in d is at position 3 (from right side). So we need to return last two bits of n as they are and other bits as 0, i.e., 00..010.

Now doing it is so easy, guess it….

Yes, you have guessing it right. See the below program.

C++

```
 #include<stdio.h>

// This function will return n % d.
// d must be one of: 1, 2, 4, 8, 16, 32, …
unsigned int getModulo(unsigned int n,
                       unsigned int d)
{
return ( n & (d - 1) );
}

// Driver Code
int main()
{
```

```
unsigned int n = 6;

// d must be a power of 2
unsigned int d = 4;
printf("%u moduo %u is %u", n, d, getModulo(n, d));

getchar();
return 0;
}
```

Java

```
 // Java code for Compute modulus division by
// a power-of-2-number
class GFG {

    // This function will return n % d.
    // d must be one of: 1, 2, 4, 8, 16, 32,
    static int getModulo(int n, int d)
    {
        return ( n & (d-1) );
    }

    // Driver Code
    public static void main(String[] args)
    {
        int n = 6;

        /*d must be a power of 2*/
        int d = 4;

        System.out.println(n+" moduo " + d +
                    " is " + getModulo(n, d));
    }
}

// This code is contributed
// by Smitha Dinesh Semwal.
```

Python3

```
 # Python code to demonstrate
# modulus division by power of 2


# This function will
# return n % d.
```

```
# d must be one of:
# 1, 2, 4, 8, 16, 32, …
def getModulo(n, d):

    return ( n & (d-1) )

# Driver program to
# test above function
n = 6

#d must be a power of 2
d = 4
print(n,"moduo",d,"is",
      getModulo(n, d))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# code for Compute modulus
// division by a power-of-2-number
using System;

class GFG {

// This function will return n % d.
// d must be one of: 1, 2, 4, 8, 16, 32, …
static uint getModulo( uint n, uint d)
{
return ( n & (d-1) );
}

// Driver code
static public void Main ()
   {
    uint n = 6;
    uint d = 4; /*d must be a power of 2*/

    Console.WriteLine( n + " moduo " + d +
                " is " + getModulo(n, d));

    }
}
// This code is contributed by vt_m.
```

PHP

```
 <?php
// This function will return n % d.
// d must be one of: 1, 2, 4, 8, 16, 32, …
function getModulo($n, $d)
{
return ( $n & ($d - 1) );
}

// Driver Code
$n = 6;

// d must be a power of 2
$d = 4;
echo $n ," moduo"," ", $d, " is ",
         " ",getModulo($n, $d);

// This code is contributed by vt_m.
?>
```

References:
http://graphics.stanford.edu/~seander/bithacks.html#ModulusDivisionEasy

Improved By : vt_m

Source

https://www.geeksforgeeks.org/compute-modulus-division-by-a-power-of-2-number/

Chapter 86

Compute the integer absolute value (abs) without branching

Compute the integer absolute value (abs) without branching - GeeksforGeeks

We need not to do anything if a number is positive. We want to change only negative numbers. Since negative numbers are stored in 2's complement form, to get the absolute value of a negative number we have to toggle bits of the number and add 1 to the result.

For example -2 in a 8 bit system is stored as follows 1 1 1 1 1 1 1 0 where leftmost bit is the sign bit. To get the absolute value of a negative number, we have to toggle all bits and add 1 to the toggled number i.e, 0 0 0 0 0 0 0 1 + 1 will give the absolute value of 1 1 1 1 1 1 1 0. Also remember, we need to do these operations only if the number is negative (sign bit is set).

Method 1
1) Set the mask as right shift of integer by 31 (assuming integers are stored using 32 bits).

```
mask = n>>31
```

2) For negative numbers, above step sets mask as 1 1 1 1 1 1 1 1 and 0 0 0 0 0 0 0 0 for positive numbers. Add the mask to the given number.

```
mask + n
```

3) XOR of mask +n and mask gives the absolute value.

```
(mask + n)^mask
```

Implementation:

```
 #include <stdio.h>
#define CHAR_BIT 8

/* This function will return absoulte value of n*/
unsigned int getAbs(int n)
{
  int const mask = n >> (sizeof(int) * CHAR_BIT - 1);
  return ((n + mask) ^ mask);
}

/* Driver program to test above function */
int main()
{
  int n = -6;
  printf("Absoute value of %d is %u", n, getAbs(n));

  getchar();
  return 0;
}
```

Method 2:
1) Set the mask as right shift of integer by 31 (assuming integers are stored using 32 bits).

```
 mask = n>>31
```

2) XOR the mask with number

```
 mask ^ n
```

3) Subtract mask from result of step 2 and return the result.

```
 (mask^n) - mask
```

Implementation:

```
 /* This function will return absoulte value of n*/
unsigned int getAbs(int n)
{
  int const mask = n >> (sizeof(int) * CHAR_BIT - 1);
  return ((n ^ mask) - mask);
}
```

On machines where branching is expensive, the above expression can be faster than the obvious approach, r = (v < 0) ? -(unsigned)v : v, even though the number of operations is the same. Please see thisfor more details about the above two methods.

References:
http://graphics.stanford.edu/~seander/bithacks.html#IntegerAbs

Source

https://www.geeksforgeeks.org/compute-the-integer-absolute-value-abs-without-branching/

Chapter 87

Compute the minimum or maximum of two integers without branching

Compute the minimum or maximum of two integers without branching - GeeksforGeeks

On some rare machines where branching is expensive, the below obvious approach to find minimum can be slow as it uses branching.

```
 /* The obvious approach to find minimum (involves branching) */
int min(int x, int y)
{
  return (x < y) ? x : y
}
```

Below are the methods to get minimum(or maximum) without using branching. Typically, the obvious approach is best, though.

Method 1(Use XOR and comparison operator)

Minimum of x and y will be

```
y ^ ((x ^ y) & -(x < y))
```

It works because if x < y, then -(x = y, then -(x < y) will be all zeros, so r = y ^ ((x ^ y) & 0) = y. On some machines, evaluating (x < y) as 0 or 1 requires a branch instruction, so there may be no advantage.

To find the maximum, use

```
x ^ ((x ^ y) & -(x < y));
```

C

```
 #include<stdio.h>

/*Function to find minimum of x and y*/
int min(int x, int y)
{
  return y ^ ((x ^ y) & -(x < y));
}

/*Function to find maximum of x and y*/
int max(int x, int y)
{
  return x ^ ((x ^ y) & -(x < y));
}

/* Driver program to test above functions */
int main()
{
  int x = 15;
  int y = 6;
  printf("Minimum of %d and %d is ", x, y);
  printf("%d", min(x, y));
  printf("\nMaximum of %d and %d is ", x, y);
  printf("%d", max(x, y));
  getchar();
}
```

Python3

```
 # Function to find minimum of x and y

def min(x, y):

    return y ^ ((x ^ y) & -(x < y))


# Function to find maximum of x and y
def max(x, y):

    return x ^ ((x ^ y) & -(x < y))


# Driver program to test above functions
x = 15
y = 6
print("Minimum of", x, "and", y, "is", end=" ")
```

```
print(min(x, y))
print("Maximum of", x, "and", y, "is", end=" ")
print(max(x, y))

# This code is contributed
# by Smitha Dinesh Semwal
```

PHP

```
 <?php
// PHP program to Compute the minimum
// or maximum of two integers without
// branching

// Function to find minimum
// of x and y
function m_in($x, $y)
{
    return $y ^ (($x ^ $y) &
                - ($x < $y));
}

// Function to find maximum
// of x and y
function m_ax($x, $y)
{
    return $x ^ (($x ^ $y) &
              - ($x < $y));
}

// Driver Code
$x = 15;
$y = 6;
echo"Minimum of"," ", $x," ","and",
    " ",$y," "," is "," ";

echo m_in($x, $y);

echo "\nMaximum of"," ",$x," ",
     "and"," ",$y," ", " is ";

echo m_ax($x, $y);

// This code is contributed by anuj_67.
?>
```

Output:

```
Minimum of 15 and 6 is 6
Maximum of 15 and 6 is 15
```

Method 2(Use subtraction and shift)
If we know that

```
INT_MIN <= (x - y) <= INT_MAX
```

, then we can use the following, which are faster because (x – y) only needs to be evaluated once.

Minimum of x and y will be

```
y + ((x - y) & ((x - y) >>(sizeof(int) * CHAR_BIT - 1)))
```

This method shifts the subtraction of x and y by 31 (if size of integer is 32). If (x-y) is smaller than 0, then (x -y)>>31 will be 1. If (x-y) is greater than or equal to 0, then (x -y)>>31 will be 0.
So if x >= y, we get minimum as y + (x-y)&0 which is y.
If x < y, we get minimum as y + (x-y)&1 which is x.

Similarly, to find the maximum use

```
x - ((x - y) & ((x - y) >> (sizeof(int) * CHAR_BIT - 1)))
```

```
 #include<stdio.h>
#define CHAR_BIT 8

/*Function to find minimum of x and y*/
int min(int x, int y)
{
  return  y + ((x - y) & ((x - y) >>
            (sizeof(int) * CHAR_BIT - 1)));
}

/*Function to find maximum of x and y*/
int max(int x, int y)
{
  return x - ((x - y) & ((x - y) >>
            (sizeof(int) * CHAR_BIT - 1)));
}

/* Driver program to test above functions */
int main()
{
```

```
  int x = 15;
  int y = 6;
  printf("Minimum of %d and %d is ", x, y);
  printf("%d", min(x, y));
  printf("\nMaximum of %d and %d is ", x, y);
  printf("%d", max(x, y));
  getchar();
}
```

Note that the 1989 ANSI C specification doesn't specify the result of signed right-shift, so above method is not portable. If exceptions are thrown on overflows, then the values of x and y should be unsigned or cast to unsigned for the subtractions to avoid unnecessarily throwing an exception, however the right-shift needs a signed operand to produce all one bits when negative, so cast to signed there.

Source:
http://graphics.stanford.edu/~seander/bithacks.html#IntegerMinOrMax

Improved By : vt_m

Source

https://www.geeksforgeeks.org/compute-the-minimum-or-maximum-max-of-two-integers-without-branching/

Chapter 88

Compute the parity of a number using XOR and table look-up

Compute the parity of a number using XOR and table look-up - GeeksforGeeks

Parity of a number refers to whether it contains an odd or even number of 1-bits. The number has "odd parity", if it contains odd number of 1-bits and is "even parity" if it contains even number of 1-bits.

```
1 --> parity of the set is odd
0 --> parity of the set is even
```

Examples:

```
Input : 254
Output : Odd Parity
Explanation : Binary of 254 is 11111110.
There are 7 ones. Thus, parity is odd.

Input : 1742346774
Output : Even
```

Method 1 : (Naive approach)
We have already discussed this method here.

Method 2 : (Efficient)
Pr-requisites : Table look up, X-OR magic

If we break a number S into two parts S_1 and S_2 such $\mathbf{S = S_1S_2}$. If we know parity of S_1 and S_2, we can compute parity of S using below facts :

1. If S_1 and S_2 have the same parity, i.e. they both have an even number of bits or an odd number of bits, their union S will have an even number of bits.
2. Therefore parity of S is XOR of parities of S_1 and S_2

The idea is to create a look up table to store parities of all 8 bit numbers. Then compute parity of whole number by dividing it into 8 bit numbers and using above facts.

Steps:

```
1. Create a look-up table for 8-bit numbers ( 0 to 255 )
   Parity of 0 is 0.
   Parity of 1 is 1.
   .
   .
   .
   Parity of 255 is 0.
2. Break the number into 8-bit chunks
   while performing XOR operations.
3. Check for the result in the table for
    the 8-bit number.
```

Since a 32 bit or 64 bit number contains constant number of bytes, the above steps take O(1) time.

Example :

```
1. Take 32-bit number : 1742346774

2. Calculate Binary of the number :
   01100111110110100001101000010110

3. Split the 32-bit binary representation into
  16-bit chunks :
0110011111011010 | 0001101000010110

4. Compute X-OR :
  0110011111011010
^ 0001101000010110
--------------------
= 0111110111001100

5. Split the 16-bit binary representation
   into 8-bit chunks : 01111101 | 11001100

6. Again, Compute X-OR :
  01111101
```

```
^ 11001100
-------------------
= 10110001
10110001 is 177 in decimal. Check
 for its parity in look-up table :
Even number of 1 = Even parity.

Thus, Parity of 1742346774 is even.
```

Below is the implementation that **works for both 32 bit and 64 bit** numbers.

C++

```
 // CPP program to illustrate Compute the parity of a
// number using XOR
#include <bits/stdc++.h>

// Generating the look-up table while pre-processing
#define P2(n) n, n ^ 1, n ^ 1, n
#define P4(n) P2(n), P2(n ^ 1), P2(n ^ 1), P2(n)
#define P6(n) P4(n), P4(n ^ 1), P4(n ^ 1), P4(n)
#define LOOK_UP P6(0), P6(1), P6(1), P6(0)

// LOOK_UP is the macro expansion to generate the table
unsigned int table[256] = { LOOK_UP };

// Function to find the parity
int Parity(int num)
{
    // Number is considered to be of 32 bits
    int max = 16;

    // Dividing the number into 8-bit
    // chunks while performing X-OR
    while (max >= 8) {
        num = num ^ (num >> max);
        max = max / 2;
    }

    // Masking the number with 0xff (11111111)
    // to produce valid 8-bit result
    return table[num & 0xff];
}

// Driver code
int main()
{
    unsigned int num = 1742346774;
```

```
    // Result is 1 for odd parity, 0 for even parity
    bool result = Parity(num);

    // Printing the desired result
    result ? std::cout << "Odd Parity" :
             std::cout << "Even Parity";

    return 0;
}
```

PHP

```
 <?php
// PHP program to illustrate
// Compute the parity of a
// number using XOR

/* Generating the look-up
table while pre-processing
#define P2(n) n, n ^ 1, n ^ 1, n
#define P4(n) P2(n), P2(n ^ 1),
              P2(n ^ 1), P2(n)
#define P6(n) P4(n), P4(n ^ 1),
              P4(n ^ 1), P4(n)
#define LOOK_UP P6(0), P6(1),
                P6(1), P6(0)

LOOK_UP is the macro expansion
to generate the table
$table = array(LOOK_UP );
*/

// Function to find
// the parity
function Parity($num)
{
    global $table;

    // Number is considered
    // to be of 32 bits
    $max = 16;

    // Dividing the number
    // into 8-bit chunks
    // while performing X-OR
    while ($max >= 8)
    {
```

```
        $num = $num ^ ($num >> $max);
        $max = (int)$max / 2;
    }

    // Masking the number with
    // 0xff (11111111) to produce
    // valid 8-bit result
    return $table[$num & 0xff];
}

// Driver code
$num = 1742346774;

// Result is 1 for odd
// parity, 0 for even parity
$result = Parity($num);

// Printing the desired result
if($result == true)
        echo "Odd Parity" ;
    else
        echo"Even Parity";

// This code is contributed by ajit
?>
```

Output:

```
Even Parity
```

Time Complexity : O(1). Note that a 32 bit or 64 bit number has fixed number of bytes (4 in case of 32 bits and 8 in case of 64 bits).

Improved By : jit_t

Source

https://www.geeksforgeeks.org/compute-parity-number-using-xor-table-look/

Chapter 89

Computing INT_MAX and INT_MIN with Bitwise operations

Computing INT_MAX and INT_MIN with Bitwise operations - GeeksforGeeks

Prerequisites :
INT_MAX and INT_MIN in C/C++ and Applications.
Arithmetic shift vs Logical shift

Suppose you have a 32-bit system :
The INT_MAX would be **01111111111111111111111111111111** and INT_MIN would be **10000000000000000000000000000000**. 0 & 1 in most-significant bit position representing the sign bit respectively.

Computing INT_MAX and INT_MIN In C/C++ :
The number 0 is represented as **000...000**(32 times).

- We compute the **NOT** of 0 to get a number with 32 1s. This number is not equal to INT_MAX because the sign bit is 1, i.e. negative number.
- Now, a right shift of this number will produce **011...111** which is INT_MAX.
- INT_MIN is NOT of INT_MAX.

Note :
0 should be taken as unsigned int.

Reason :
If 0 is signed, during Step 2, right shift of 111..111 will yield 111…111. This is because arithmetic right shift preserves the sign of the number.
In Java, we have the feature of logical right shift available to us.

C/C++

```
 // CPP code to compute INT_MAX and INT_MIN using
// bitwise operations
#include <bits/stdc++.h>
using namespace std;

void printMinMaxValues()
{
    // 0 saved as unsigned int
    unsigned int max = 0;

    // Computing NOT of 0
    max = ~max;

    // 1 time arithmetic right shift
    max = max >> 1;

    // Computing INT_MIN
    int min = max;

    // INT_MIN = ~INT_MAX
    min = ~min;

    // Printing the result
    cout << "INT_MAX : " << max
         << " INT_MIN : " << min;
}

// Driver code
int main()
{
    printMinMaxValues();
    return 0;
}
```

Java

```
 // Java code to compute INT_MAX and INT_MIN using
// bitwise operations
public class Solution
{
    static void printMinMaxValues()
    {
        int max = 0;

        // Computing NOT of 0
        max = ~max;

        // 1 time logical right shift for INT_MAX
```

```
        max = max >>> 1;

        // Computing INT_MIN
        int min = max;

        // INT_MIN = ~INT_MAX
        min = ~max;

        // Printing the result
        System.out.println("INT_MAX " + max +
                           " INT_MIN " + min);
    }

    public static void main(String[] args)
    {
        printMinMaxValues();
    }
}
```

Output:

```
INT_MAX 2147483647 INT_MIN -2147483648
```

Asked in : Google

Source

https://www.geeksforgeeks.org/computing-int_max-int_min-bitwise-operations/

Chapter 90

Construct an array from XOR of all elements of array except element at same index

Construct an array from XOR of all elements of array except element at same index - GeeksforGeeks

Given an array A[] having n positive elements. The task to create another array B[] such as B[i] is XOR of all elements of array A[] except A[i].

Examples :

```
Input : A[] = {2, 1, 5, 9}
Output : B[] = {13, 14, 10, 6}

Input : A[] = {2, 1, 3, 6}
Output : B[] = {4, 7, 5, 0}
```

Naive Approach :
We can simple calculate B[i] as XOR of all elements of A[] except A[i], as

```
for (int i = 0; i < n; i++)
{
    B[i] = 0;
    for (int j = 0; j < n; j++)
        if ( i != j)
            B[i] ^= A[j];
}
```

Time complexity for this naive approach is O (n^2).
Auxiliary Space for this naive approach is O (n).

Optimized Approach :
First calculate XOR of all elements of array A[] say 'xor', and for each element of array A[] calculate **A[i] = xor ^ A[i]** .

```
int xor = 0;
for (int i = 0; i < n; i++)
    xor ^= A[i];

for (int i = 0; i < n; i++)
        A[i] = xor ^ A[i];
```

Time complexity for this approach is O (n).
Auxiliary Space for this approach is O (1).

C++

```
 // C++ program to construct array from
// XOR of elements of given array
#include <bits/stdc++.h>
using namespace std;

// function to construct new array
void constructXOR(int A[], int n)
{
    // calculate xor of array
    int XOR = 0;
    for (int i = 0; i < n; i++)
        XOR ^= A[i];

    // update array
    for (int i = 0; i < n; i++)
        A[i] = XOR ^ A[i];
}

// Driver code
int main()
{
    int A[] = { 2, 4, 1, 3, 5};
    int n = sizeof(A) / sizeof(A[0]);
    constructXOR(A, n);

    // print result
```

```
    for (int i = 0; i < n; i++)
        cout << A[i] << " ";
    return 0;
}
```

Java

```
 // Java program to construct array from
// XOR of elements of given array
class GFG
{

    // function to construct new array
    static void constructXOR(int A[], int n)
    {

        // calculate xor of array
        int XOR = 0;
        for (int i = 0; i < n; i++)
            XOR ^= A[i];

        // update array
        for (int i = 0; i < n; i++)
            A[i] = XOR ^ A[i];
    }

    // Driver code
    public static void main(String[] args)
    {
        int A[] = { 2, 4, 1, 3, 5};
        int n = A.length;
        constructXOR(A, n);

        // print result
        for (int i = 0; i < n; i++)
            System.out.print(A[i] + " ");
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python 3 program to construct
# array from XOR of elements
# of given array
```

```
# function to construct new array
def constructXOR(A, n):

    # calculate xor of array
    XOR = 0
    for i in range(0, n):
        XOR ^= A[i]

    # update array
    for i in range(0, n):
        A[i] = XOR ^ A[i]

# Driver code
A = [ 2, 4, 1, 3, 5 ]
n = len(A)
constructXOR(A, n)

# print result
for i in range(0,n):
    print(A[i], end =" ")

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to construct array from
// XOR of elements of given array
using System;

class GFG
{

    // function to construct new array
    static void constructXOR(int []A, int n)
    {

        // calculate xor of array
        int XOR = 0;
        for (int i = 0; i < n; i++)
            XOR ^= A[i];

        // update array
        for (int i = 0; i < n; i++)
            A[i] = XOR ^ A[i];
    }

    // Driver code
    public static void Main()
```

```
    {
        int []A = { 2, 4, 1, 3, 5};
        int n = A.Length;
        constructXOR(A, n);

        // print result
        for (int i = 0; i < n; i++)
        Console.Write(A[i] + " ");
    }
}

// This code is contributed by nitin mittal
```

Output:

```
3 5 0 2 4
```

Related Problem :
A Product Array Puzzle

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/construct-an-array-from-xor-of-all-elements-of-array-except-element-at-same-index

Chapter 91

Convert a binary number to octal

Convert a binary number to octal - GeeksforGeeks

The problem is to convert the given binary number (represented as string) to its equivalent octal number. The input could be very large and may not fit even into unsigned long long int.

Examples:

```
Input : 110001110
Output : 616

Input  : 1111001010010100001.010110110011011
Output : 1712241.26633
```

The idea is to consider the binary input as a string of characters and then follow the steps:

1. Get length of substring to the left and right of the decimal point(‘.’) as **left_len** and **right_len**.
2. If **left_len** is not a multiple of 3 add min number of 0’s in the beginning to make length of left substring a multiple of 3.
3. If **right_len** is not a multiple of 3 add min number of 0’s in the end to make length of right substring a multiple of 3.
4. Now, from the left extract one by one substrings of length 3 and add its corresponding octal code to the result.
5. If in between a decimal(‘.’) is encountered then add it to the result.

```
 // C++ implementation to convert a binary number
// to octal number
```

```
#include <bits/stdc++.h>
using namespace std;

// function to create map between binary
// number and its equivalent octal
void createMap(unordered_map<string, char> *um)
{
    (*um)["000"] = '0';
    (*um)["001"] = '1';
    (*um)["010"] = '2';
    (*um)["011"] = '3';
    (*um)["100"] = '4';
    (*um)["101"] = '5';
    (*um)["110"] = '6';
    (*um)["111"] = '7';
}

// Function to find octal equivalent of binary
string convertBinToOct(string bin)
{
    int l = bin.size();
    int t = bin.find_first_of('.');

    // length of string before '.'
    int len_left = t != -1 ? t : l;

    // add min 0's in the beginning to make
    // left substring length divisible by 3
    for (int i = 1; i <= (3 - len_left % 3) % 3; i++)
        bin = '0' + bin;

    // if decimal point exists
    if (t != -1)
    {
        // length of string after '.'
        int len_right = l - len_left - 1;

        // add min 0's in the end to make right
        // substring length divisible by 3
        for (int i = 1; i <= (3 - len_right % 3) % 3; i++)
            bin = bin + '0';
    }

    // create map between binary and its
    // equivalent octal code
    unordered_map<string, char> bin_oct_map;
    createMap(&bin_oct_map);
```

```
    int i = 0;
    string octal = "";

    while (1)
    {
        // one by one extract from left, substring
        // of size 3 and add its octal code
        octal += bin_oct_map[bin.substr(i, 3)];
        i += 3;
        if (i == bin.size())
            break;

        // if '.' is encountered add it to result
        if (bin.at(i) == '.')
        {
            octal += '.';
            i++;
        }
    }

    // required octal number
    return octal;
}

// Driver program to test above
int main()
{
    string bin = "1111001010010100001.010110110011011";
    cout << "Octal number = "
         << convertBinToOct(bin);
    return 0;
}
```

Output:

```
Octal number = 1712241.26633
```

Time Complexity: O(n), where n is the length of string.

Source

https://www.geeksforgeeks.org/convert-binary-number-octal/

Chapter 92

Convert decimal fraction to binary number

Convert decimal fraction to binary number - GeeksforGeeks

Given an fraction decimal number n and integer k, convert decimal number n into equivalent binary number up-to k precision after decimal point.

```
Input: n = 2.47, k = 5
Output: 10.01111

Input: n = 6.986 k = 8
Output: 110.11111100
```

We strongly recommend that you click here and practice it, before moving on to the solution.

A) Convert the integral part of decimal to binary equivalent

1. Divide the decimal number by 2 and store remainders in array.
2. Divide the quotient by 2.
3. Repeat step 2 until we get the quotient equal to zero.
4. Equivalent binary number would be reverse of all remainders of step 1.

B) Convert the fractional part of decimal to binary equivalent

1. Multiply the fractional decimal number by 2.
2. Integral part of resultant decimal number will be first digit of fraction binary number.
3. Repeat step 1 using only fractional part of decimal number and then step 2.

C) Combine both integral and fractional part of binary number.

Illustration

```
Let's take an example for n = 4.47 k = 3

Step 1: Conversion of 4 to binary
1. 4/2 : Remainder = 0 : Quotient = 2
2. 2/2 : Remainder = 0 : Quotient = 1
3. 1/2 : Remainder = 1 : Quotient = 0

So equivalent binary of integral part of decimal is 100.

Step 2: Conversion of .47 to binary
1. 0.47 * 2 = 0.94, Integral part: 0
2. 0.94 * 2 = 1.88, Integral part: 1
3. 0.88 * 2 = 1.76, Integral part: 1

So equivalent bianry of fractional part of decimal is .011

Step 3: Combined the result of step 1 and 2.

Final answer can be written as:
100 + .011 = 100.011
```

C++ Program to demonstrate above steps:

```
 // C++ program to convert fractional decimal
// to binary number
#include<bits/stdc++.h>
using namespace std;

// Function to convert decimal to binary upto
// k-precision after decimal point
string decimalToBinary(double num, int k_prec)
{
    string binary = "";

    // Fetch the integral part of decimal number
    int Integral = num;

    // Fetch the fractional part decimal number
    double fractional = num - Integral;

    // Conversion of integral part to
    // binary equivalent
    while (Integral)
    {
        int rem = Integral % 2;

        // Append 0 in binary
```

```
        binary.push_back(rem +'0');

        Integral /= 2;
    }

    // Reverse string to get original binary
    // equivalent
    reverse(binary.begin(),binary.end());

    // Append point before conversion of
    // fractional part
    binary.push_back('.');

    // Conversion of fractional part to
    // binary equivalent
    while (k_prec--)
    {
        // Find next bit in fraction
        fractional *= 2;
        int fract_bit = fractional;

        if (fract_bit == 1)
        {
            fractional -= fract_bit;
            binary.push_back(1 + '0');
        }
        else
            binary.push_back(0 + '0');
    }

    return binary;
}

// Driver code
int main()
{

    double n = 4.47;
    int k = 3;
    cout << decimalToBinary(n, k) << "\n";

    n = 6.986 , k = 5;
    cout << decimalToBinary(n, k);
    return 0;
}
```

Output:

```
100.011
110.11111
```

Time complexity: O(len(n))
Auxiliary space: O(len(n))
where len() is the total digits contain in number n.

Convert Binary fraction to Decimal

Reference:
http://cs.furman.edu/digitaldomain/more/ch6/dec_frac_to_bin.htm
http://www.cquestions.com/2011/07/c-program-for-fractional-decimal-to.html

Source

https://www.geeksforgeeks.org/convert-decimal-fraction-binary-number/

Chapter 93

Copy set bits in a range

Copy set bits in a range - GeeksforGeeks

Given two numbers x and y, and a range [l, r] where 1 <= l, r <= 32. The task is consider set bits of y in range [l, r] and set these bits in x also. Examples :

```
Input  : x = 10, y = 13, l = 2, r = 3
Output : x = 14
Binary representation of 10 is 1010 and
that of y is 1101. There is one set bit
in y at 3'rd position (in given range).
After we copy this bit to x, x becomes 1110
which is binary representation of 14.

Input  : x = 8, y = 7, l = 1, r = 2
Output : x = 11
```

Source : D E Shaw Interview

Method 1 (One by one copy bits)
We can one by one find set bits of y by traversing given range. For every set bit, we OR it to existing bit of x, so that the becomes set in x, if it was not set. Below is C++ implementation.

```
 // C++ program to rearrange array in alternating
// C++ program to copy set bits in a given
// range [l, r] from y to x.
#include <bits/stdc++.h>
using namespace std;

// Copy set bits in range [l, r] from y to x.
// Note that x is passed by reference and modified
```

```
// by this function.
void copySetBits(unsigned &x, unsigned y,
                 unsigned l, unsigned r)
{
   // l and r must be between 1 to 32
   // (assuming ints are stored using
   //  32 bits)
   if (l < 1 || r > 32)
      return ;

   // Travers in given range
   for (int i=l; i<=r; i++)
   {
       // Find a mask (A number whose
       // only set bit is at i'th position)
       int mask = 1 << (i-1);

       // If i'th bit is set in y, set i'th
       // bit in x also.
       if (y & mask)
          x = x | mask;
   }
}

// Driver code
int main()
{
   unsigned x = 10, y = 13, l = 2, r = 3;
   copySetBits(x, y, l, r);
   cout << "Modified x is " << x;
   return 0;
}
```

Output :

```
Modified x is 14
```

Method 2 (Copy all bits using one bit mask)

```
 // C++ program to copy set bits in a given
// range [l, r] from y to x.
#include <bits/stdc++.h>
using namespace std;

// Copy set bits in range [l, r] from y to x.
// Note that x is passed by reference and modified
```

```
// by this function.
void copySetBits(unsigned &x, unsigned y,
                 unsigned l, unsigned r)
{
    // l and r must be between 1 to 32
    if (l < 1 || r > 32)
        return ;

    // get the length of the mask
    int maskLength = (1<<(r-l+1)) - 1;

    // Shift the mask to the required position
    // "&" with y to get the set bits at between
    // l ad r in y
    int mask = ((maskLength)<<(l-1)) & y ;
    x = x | mask;
}

// Driver code
int main()
{
   unsigned x = 10, y = 13, l = 2, r = 3;
   copySetBits(x, y, l, r);
   cout << "Modified x is " << x;
   return 0;
}
```

Output :

```
Modified x is 14
```

Thanks to Ashish Rathi for suggesting this solution in a comment.

This article is contributed by **Rishi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/copy-set-bits-in-a-range/

Chapter 94

Count all pairs of an array which differ in K bits

Count all pairs of an array which differ in K bits - GeeksforGeeks

Given an array of size n and integer k, count all pairs in array which differ in exactly K bits of binary representation of both the numbers.

The input arrays have elements with small values and possibly many repetitions.

```
Input: arr[] = {2, 4, 1, 3, 1}
       k = 2
Output: 5
Explanation:
There are only 4 pairs which differs in
exactly 2 bits of binary representation:
(2, 4), (1, 2) [Two times] and (4, 1)
[Two times]

Input  : arr[] = {2, 1, 2, 1}
         k = 2
Output :  4
```

We strongly recommend that you click here and practice it, before moving on to the solution.

Naive Approach

A brute force is to run the two loops one inside the another and select the pairs one by one and take a XOR of both elements. The result of XORed value contains a set bits which are differ in both the elements. Now we just need to count total set bits so that we compare it with value K.

C++

```
 // C++ program to count all pairs with bit difference
// as k
#include<bits/stdc++.h>
using namespace std;

// Utility function to count total ones in a number
int bitCount(int n)
{
    int count = 0;
    while (n)
    {
        if (n & 1)
            ++count;
        n >>= 1;
    }
    return count;
}

// Function to count pairs of K different bits
long long countPairsWithKDiff(int arr[], int n, int k)
{
    long long ans = 0; // initialize final answer

    for (int i = 0; i < n-1; ++i)
    {
        for (int j = i + 1; j < n; ++j)
        {
            int xoredNum = arr[i] ^ arr[j];

            // Check for K differ bit
            if (k == bitCount(xoredNum))
                ++ans;
        }
    }
    return ans;
}

// Driver code
int main()
{
    int k = 2;
    int arr[] = {2, 4, 1, 3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Total pairs for k = " << k << " are = "
         << countPairsWithKDiff(arr, n, k) << "\n";

    return 0;
```

```
}
```

PHP

```
 <?php
// PHP program to count all
// pairs with bit difference
// as k

// Utility function to count
// total ones in a number
function bitCount($n)
{
    $count = 0;
    while ($n)
    {
        if ($n & 1)
            ++$count;
        $n >>= 1;
    }
    return $count;
}

// Function to count pairs
// of K different bits
function countPairsWithKDiff($arr, $n, $k)
{

    // initialize final answer
    $ans = 0;

    for ($i = 0; $i < $n-1; ++$i)
    {
        for ($j = $i + 1; $j < $n; ++$j)
        {
            $xoredNum = $arr[$i] ^ $arr[$j];

            // Check for K differ bit
            if ($k == bitCount($xoredNum))
                ++$ans;
        }
    }
    return $ans;
}

    // Driver code
    $k = 2;
    $arr = array(2, 4, 1, 3, 1);
```

```
    $n = count($arr);

    echo "Total pairs for k = " , $k , " are = "
        , countPairsWithKDiff($arr, $n, $k) , "\n";

// This code is contributed by anuj_67.
?>
```

```
Output:Total pairs for k = 2 are = 5
```

Time complexity: O(N^2 * log MAX) where MAX is maximum element in input array.
Auxiliary space: O(1)

Efficient approach

This approach is efficient for the cases when input array has small elements and possibly many repetitions. The idea is to iterate from 0 to max(arr[i]) and for every pair(i, j) check the number of set bits in (i ^ j) and compare this with K. We can use inbuilt function of gcc(__builtin_popcount) or precompute such array to make the check faster. If number of ones in i ^ j is equals to K then we will add the total count of both i and j.

```
 // Below is C++ approach of finding total k bit
// difference pairs
#include<bits/stdc++.h>
using namespace std;

// Function to calclate K bit different pairs in array
long long kBitDifferencePairs(int arr[], int n, int k)
{
    // Get the maximum value among all array elemensts
    int MAX = *max_element(arr, arr+n);

    // Set the count array to 0, count[] stores the
    // total frequency of array elements
    long long count[MAX+1];
    memset(count, 0, sizeof(count));

    for (int i=0; i < n; ++i)
        ++count[arr[i]];

    // Initialize result
    long long ans = 0;

    // For 0 bit answer will be total count of same number
```

```
    if (k == 0)
    {
        for (int i = 0; i <= MAX; ++i)
            ans += (count[i] * (count[i] - 1)) / 2;

        return ans;
    }


    for (int i = 0; i <= MAX; ++i)
    {
        // if count[i] is 0, skip the next loop as it
        // will not contribute the answer
        if (!count[i])
           continue;

        for (int j = i + 1; j <= MAX; ++j)
        {
            //Update answer if k differ bit found
            if ( __builtin_popcount(i ^ j) == k)
                ans += count[i] * count[j];
        }
    }
    return ans;
}

// Driver code
int main()
{
    int k = 2;
    int arr[] = {2, 4, 1, 3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Total pairs for k = " << k << " are = "
         << kBitDifferencePairs(arr, n, k) << "\n";

    k = 3;
    cout << "Total pairs for k = " << k << " are = "
         << kBitDifferencePairs(arr, n, k) ;
    return 0;
}


Output:Total pairs for k = 2 are = 5
```

Time complexity: O(MAX^2) where MAX is maximum element in input array.
Auxiliary space: O(MAX)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/count-all-pairs-of-an-array-which-differ-in-k-bits/

Chapter 95

Count all pairs with given XOR

Count all pairs with given XOR - GeeksforGeeks

Given an array of distinct positive integers and a number x, find the number of pairs of integers in the array whose XOR is equal to x.
Examples:

```
Input : arr[] = {5, 4, 10, 15, 7, 6}, x = 5
Output : 1
Explanation :  (10 ^ 15) = 5

Input : arr[] = {3, 6, 8, 10, 15, 50}, x = 5
Output : 2
Explanation : (3 ^ 6) = 5 and (10 ^ 15) = 5
```

A **Simple solution** is to traverse each element and check if there's another number whose XOR with it is equal to x. This solution takes $O(n^2)$ time.

An **efficient solution** of this problem take O(n) time. The idea is based on the fact that arr[i] ^ arr[j] is equal to x if and only if arr[i] ^ x is equal to arr[j].

```
1) Initialize result as 0.
2) Create an empty hash set "s".
3) Do following for each element arr[i] in arr[]
   (a)    If x ^ arr[i] is in "s", then increment result by 1.
   (b)    Insert arr[i] into the hash set "s".
3) return result.
```

C++

```
 // C++ program to Count all pair with given XOR
// value x
#include<bits/stdc++.h>

using namespace std;

// Returns count of pairs in arr[0..n-1] with XOR
// value equals to x.
int xorPairCount(int arr[], int n, int x)
{
    int result = 0; // Initialize result

    // create empty set that stores the visiting
    // element of array.
    // Refer below post for details of unordered_set
    // https://www.geeksforgeeks.org/unorderd_set-stl-uses/
    unordered_set<int> s;

    for (int i=0; i<n ; i++)
    {
        // If there exist an element in set s
        // with XOR equals to x^arr[i], that means
        // there exist an element such that the
        // XOR of element with arr[i] is equal to
        // x, then increment count.
        if (s.find(x^arr[i]) != s.end())
            result++;

        // Make element visited
        s.insert(arr[i]);
    }

    // return total count of pairs with XOR equal to x
    return result;
}

// driver program
int main()
{
    int arr[] = {5 , 4 ,10, 15, 7, 6};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 5;
    cout << "Count of pairs with given XOR = "
         << xorPairCount(arr, n, x);
    return 0;
}
```

Python3

```
 #Python3 program to count all the pair with given xor

#Returns count of pairs in arr[0..n-1] with XOR
#value equals to x.
def xorPairCount(arr,n,x):
    result = 0 #Initialize result
    #create empty set that stores the visiting
    #element of array.
    s = set()
    for i in range(0,n):
        #If there exist an element in set s
        #with XOR equals to x^arr[i], that means
        #there exist an element such that the
        #XOR of element with arr[i] is equal to
        #x, then increment count.
        if(x^arr[i] in s):
            result = result +1
        #Make element visited
        s.add(arr[i])
    return result
# Driver Code

if __name__ == "__main__":
    arr = [5, 4, 10, 15, 7, 6]
    n = len(arr)
    x = 5
    print("Count of pair with given XOR = "
         +str(xorPairCount(arr,n,x)))

#program by Anubhav Natani
```

Output:

```
Count of pairs with given XOR = 1
```

Time complexity : O(n)

How to handle duplicates?
The above efficient solution doesn't work if there are duplicates in input array. For example, the above solution produces different results for {2, 2, 5} and {5, 2, 2}. To handle duplicates, we store counts of occurrences of all elements. We use unordered_map instead of unordered_set.

C++

```
 // C++ program to Count all pair with given XOR
```

```
// value x
#include<bits/stdc++.h>

using namespace std;

// Returns count of pairs in arr[0..n-1] with XOR
// value equals to x.
int xorPairCount(int arr[], int n, int x)
{
    int result = 0; // Initialize result

    // create empty map that stores counts of
    // individual elements of array.
    unordered_map<int, int> m;

    for (int i=0; i<n ; i++)
    {
        int curr_xor =  x^arr[i];

        // If there exist an element in map m
        // with XOR equals to x^arr[i], that means
        // there exist an element such that the
        // XOR of element with arr[i] is equal to
        // x, then increment count.
        if (m.find(curr_xor) != m.end())
            result += m[curr_xor];

        // Increment count of current element
        m[arr[i]]++;
    }


    // return total count of pairs with XOR equal to x
    return result;
}

// driver program
int main()
{
    int arr[] = {2, 5, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 0;
    cout << "Count of pairs with given XOR = "
         << xorPairCount(arr, n, x);
    return 0;
}
```

Output:

```
Count of pairs with given XOR = 1
```

Time complexity : O(n)

Improved By : NerdCaps, AnubhavNatani

Source

https://www.geeksforgeeks.org/count-pairs-given-xor/

Chapter 96

Count inversions in an array | Set 3 (Using BIT)

Count inversions in an array | Set 3 (Using BIT) - GeeksforGeeks

Inversion Count for an array indicates – how far (or close) the array is from being sorted. If array is already sorted then inversion count is 0. If array is sorted in reverse order that inversion count is the maximum.

```
Two elements a[i] and a[j] form an inversion if
a[i] > a[j] and i < j. For simplicity, we may
assume that all elements are unique.

Example:
Input:  arr[] = {8, 4, 2, 1}
Output: 6
Given array has six inversions (8,4), (4,2),
(8,2), (8,1), (4,1), (2,1).
```

We have already discussed below methods to solve inversion count
1)Naive and Modified Merge Sort
2) Using AVL Tree

Background:
BIT basically supports two operations for an array arr[] of size n:

1. Sum of elements till arr[i] in O(Log n) time.
2. Update an array element in O(Log n) time.

BIT is implemented using an array and works in form of trees. Note that there are two ways of looking at BIT as a tree.

1. The sum operation where parent of index x is "x - (x & -x)".
2. The update operation where parent of index x is "x + (x & -x)".

We recommend you to refer Binary Indexed Tree (BIT) before further reading this post.

Basic Approach using BIT of size Θ(maxElement):
The idea is to iterate the array from n-1 to 0. When we are at i'th index, we check how many numbers less than arr[i] are present in BIT and add it to the result. To get the count of smaller elements, getSum() of BIT is used. In his basic idea, BIT is represented as an array of size equal to maximum element plus one. So that elements can be used as an index. After that we add current element to to the BIT[] by doing an update operation that updates count of current element from 0 to 1, and therefore updates ancestors of current element in BIT (See update() in BIT for details).

Below is C++ implementation of basic idea that uses BIT.

```
 // C++ program to count inversions using Binary Indexed Tree
#include<bits/stdc++.h>
using namespace std;

// Returns sum of arr[0..index]. This function assumes
// that the array is preprocessed and partial sums of
// array elements are stored in BITree[].
int getSum(int BITree[], int index)
{
    int sum = 0; // Initialize result

    // Traverse ancestors of BITree[index]
    while (index > 0)
    {
        // Add current element of BITree to sum
        sum += BITree[index];

        // Move index to parent node in getSum View
        index -= index & (-index);
    }
    return sum;
}

// Updates a node in Binary Index Tree (BITree) at given index
// in BITree.  The given value 'val' is added to BITree[i] and
// all of its ancestors in tree.
void updateBIT(int BITree[], int n, int index, int val)
{
    // Traverse all ancestors and add 'val'
    while (index <= n)
    {
       // Add 'val' to current node of BI Tree
       BITree[index] += val;
```

```
        // Update index to that of parent in update View
        index += index & (-index);
    }
}

// Returns inversion count arr[0..n-1]
int getInvCount(int arr[], int n)
{
    int invcount = 0; // Initialize result

    // Find maximum element in arr[]
    int maxElement = 0;
    for (int i=0; i<n; i++)
        if (maxElement < arr[i])
            maxElement = arr[i];

    // Create a BIT with size equal to maxElement+1 (Extra
    // one is used so that elements can be directly be
    // used as index)
    int BIT[maxElement+1];
    for (int i=1; i<=maxElement; i++)
        BIT[i] = 0;

    // Traverse all elements from right.
    for (int i=n-1; i>=0; i--)
    {
        // Get count of elements smaller than arr[i]
        invcount += getSum(BIT, arr[i]-1);

        // Add current element to BIT
        updateBIT(BIT, maxElement, arr[i], 1);
    }

    return invcount;
}

// Driver program
int main()
{
    int arr[] = {8, 4, 2, 1};
    int n = sizeof(arr)/sizeof(int);
    cout << "Number of inversions are : " << getInvCount(arr,n);
    return 0;
}
```

Output:

```
Number of inversions are : 6
```

Time Complexity :- The update function and getSum function runs for O(log(maximumelement)) and we are iterating over n elements. So overall time complexity is : O(nlog(maximumelement)).
Auxiliary space : O(maxElement)

Better Approach using BIT of size Θ(n):
The problem with the previous approach is that it doesn't work for negative numbers as index cannot be negative. Also by updating the value till maximum element we waste time and space as it is quite possible that we may never use intermediate value. For example, lots of space an time is wasted for an array like {1, 100000}.

The idea is to convert given array to an array with values from 1 to n and relative order of smaller and greater elements remains

```
Example :-
arr[] = {7, -90, 100, 1}

It gets  converted to,
arr[] = {3, 1, 4 ,2 }
 as -90 < 1 < 7 < 100.
```

We only have to make BIT[] of number of elements instead of maximum element.
Changing element will not have any change in the answer as the greater elements remain greater and at same position.

```
 // C++ program to count inversions using Binary Indexed Tree
#include<bits/stdc++.h>
using namespace std;

// Returns sum of arr[0..index]. This function assumes
// that the array is preprocessed and partial sums of
// array elements are stored in BITree[].
int getSum(int BITree[], int index)
{
    int sum = 0; // Initialize result

    // Traverse ancestors of BITree[index]
    while (index > 0)
    {
        // Add current element of BITree to sum
        sum += BITree[index];

        // Move index to parent node in getSum View
        index -= index & (-index);
    }
    return sum;
```

```
}

// Updates a node in Binary Index Tree (BITree) at given index
// in BITree.  The given value 'val' is added to BITree[i] and
// all of its ancestors in tree.
void updateBIT(int BITree[], int n, int index, int val)
{
    // Traverse all ancestors and add 'val'
    while (index <= n)
    {
       // Add 'val' to current node of BI Tree
       BITree[index] += val;

       // Update index to that of parent in update View
       index += index & (-index);
    }
}

// Converts an array to an array with values from 1 to n
// and relative order of smaller and greater elements remains
// same.  For example, {7, -90, 100, 1} is converted to
// {3, 1, 4 ,2 }
void convert(int arr[], int n)
{
    // Create a copy of arrp[] in temp and sort the temp array
    // in increasing order
    int temp[n];
    for (int i=0; i<n; i++)
        temp[i] = arr[i];
    sort(temp, temp+n);

    // Traverse all array elements
    for (int i=0; i<n; i++)
    {
        // lower_bound() Returns pointer to the first element
        // greater than or equal to arr[i]
        arr[i] = lower_bound(temp, temp+n, arr[i]) - temp + 1;
    }
}

// Returns inversion count arr[0..n-1]
int getInvCount(int arr[], int n)
{
    int invcount = 0; // Initialize result

     // Convert arr[] to an array with values from 1 to n and
     // relative order of smaller and greater elements remains
     // same.  For example, {7, -90, 100, 1} is converted to
```

```
    //  {3, 1, 4 ,2 }
    convert(arr, n);

    // Create a BIT with size equal to maxElement+1 (Extra
    // one is used so that elements can be directly be
    // used as index)
    int BIT[n+1];
    for (int i=1; i<=n; i++)
        BIT[i] = 0;

    // Traverse all elements from right.
    for (int i=n-1; i>=0; i--)
    {
        // Get count of elements smaller than arr[i]
        invcount += getSum(BIT, arr[i]-1);

        // Add current element to BIT
        updateBIT(BIT, n, arr[i], 1);
    }

    return invcount;
}

// Driver program
int main()
{
    int arr[] = {8, 4, 2, 1};
    int n = sizeof(arr)/sizeof(int);
    cout << "Number of inversions are : " << getInvCount(arr,n);
    return 0;
}
```

Output:

```
Number of inversions are : 6
```

Time Complexity :- The update function and getSum function runs for O(log(n)) and we are iterating over n elements. So overall time complexity is : O(nlogn).
Auxiliary space : O(n)

This article is contributed by Abhiraj Smit. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/count-inversions-array-set-3-using-bit/

Chapter 97

Count minimum bits to flip such that XOR of A and B equal to C

Count minimum bits to flip such that XOR of A and B equal to C - GeeksforGeeks

Given a sequence of three binary sequences A, B and C of N bits. Count the minimum bits required to flip in A and B such that XOR of A and B is equal to C. For **Example :**

```
Input: N = 3
       A = 110
       B = 101
       C = 001
Output: 1
We only need to flip the bit of 2nd position
of either A or B, such that A ^ B = C i.e.,
100 ^ 101 = 001
```

A **Naive approach** is to generate all possible combination of bits in A and B and then XORing them to Check whether it is equal to C or not. **Time complexity** of this approach grows exponentially so it would not be better for large value of N.

An **Efficient** approach is to use concept of XOR.

```
XOR Truth Table
  Input    Output
 X     Y     Z
 0     0  -  0
 0     1  -  1
```

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

If we generalize, we will find that at any position of A and B, we just only need to flip i^{th} (0 to N-1) position of either A or B otherwise we will not able to achieve minimum no of Bits. So at any position of i (0 to N-1) you will encounter two type of situation i.e., either A[i] == B[i] or A[i] != B[i]. Let's discuss it one by one.

- If A[i] == B[i] then XOR of these bits will be 0, two cases arise in C[]: C[i]==0 or C[i]==1.
  If C[i] == 0, then no need to flip the bit but if C[i] == 1 then we have to flip the bit either in A[i] or B[i] so that 1^0 == 1 or 0^1 == 1.
- If A[i] != B[i] then XOR of these Bits gives a 1, In C two cases again arise i.e., either C[i] == 0 or C[i] == 1.
  Therefore if C[i] == 1, then we need not to flip the bit but if C[i] == 0, then we need to flip the bit either in A[i] or B[i] so that 0^0==0 or 1^1==0

C++

```
 // C++ code to count the Minimum bits in A and B
#include<bits/stdc++.h>
using namespace std;

int totalFlips(char *A, char *B, char *C, int N)
{
    int count = 0;
    for (int i=0; i < N; ++i)
    {
        // If both A[i] and B[i] are equal
        if (A[i] == B[i] && C[i] == '1')
            ++count;

        // If Both A and B are unequal
        else if (A[i] != B[i] && C[i] == '0')
            ++count;
    }
    return count;
}

//Driver Code
int main()
{
    //N represent total count of Bits
    int N = 5;
    char a[] = "10100";
    char b[] = "00010";
    char c[] = "10011";
```

```
    cout << totalFlips(a, b, c, N);

    return 0;
}
```

Java

```
 // Java code to count the Minimum bits in A and B
class GFG {

    static int totalFlips(String A, String B,
                                String C, int N)
    {
        int count = 0;

        for (int i = 0; i < N; ++i)
        {
            // If both A[i] and B[i] are equal
            if (A.charAt(i) == B.charAt(i) &&
                           C.charAt(i) == '1')
                ++count;

            // If Both A and B are unequal
            else if (A.charAt(i) != B.charAt(i)
                          && C.charAt(i) == '0')
                ++count;
        }

        return count;
    }

    //driver code
    public static void main (String[] args)
    {
        //N represent total count of Bits
        int N = 5;
        String a = "10100";
        String b = "00010";
        String c = "10011";

        System.out.print(totalFlips(a, b, c, N));
    }
}

// This code is contributed by Anant Agarwal.
```

Python

```
 # Python code to find minimum bits to be flip
def totalFlips(A, B, C, N):

    count = 0
    for i in range(N):

        # If both A[i] and B[i] are equal
        if A[i] == B[i] and C[i] == '1':
            count=count+1

        # if A[i] and B[i] are unequal
        elif A[i] != B[i] and C[i] == '0':
            count=count+1
    return count


# Driver Code
# N represent total count of Bits
N = 5
a = "10100"
b = "00010"
c = "10011"
print(totalFlips(a, b, c, N))
```

C#

```
 // C# code to count the Minimum
// bits flip in A and B
using System;

class GFG {

    static int totalFlips(string A, string B,
                          string C, int N)
    {
        int count = 0;
        for (int i = 0; i < N; ++i) {

            // If both A[i] and B[i] are equal
            if (A[i] == B[i] && C[i] == '1')
                ++count;

            // If Both A and B are unequal
            else if (A[i] != B[i] && C[i] == '0')
                ++count;
        }
        return count;
    }
```

```
    // Driver code
    public static void Main()
    {
        // N represent total count of Bits
        int N = 5;
        string a = "10100";
        string b = "00010";
        string c = "10011";

        Console.Write(totalFlips(a, b, c, N));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP code to count the
// Minimum bits in A and B

function totalFlips($A, $B, $C, $N)
{
    $count = 0;
    for ($i = 0; $i < $N; ++$i)
    {
        // If both A[i] and
        // B[i] are equal
        if ($A[$i] == $B[$i] &&
            $C[$i] == '1')
            ++$count;

        // If Both A and
        // B are unequal
        else if ($A[$i] != $B[$i] &&
                 $C[$i] == '0')
            ++$count;
    }
    return $count;
}

// Driver Code

// N represent total count of Bits
$N = 5;
$a = "10100";
$b = "00010";
```

```
$c = "10011";

echo totalFlips($a, $b, $c, $N);

// This code is contributed by nitin mittal.
?>
```

Output:

```
2
```

Time Complexity: O(N)
Auxiliary space: O(1)

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/count-minimum-bits-flip-xor-b-equal-c/

Chapter 98

Count number of bits to be flipped to convert A to B

Count number of bits to be flipped to convert A to B - GeeksforGeeks

Given two numbers 'a' and b'. Write a program to count number of bits needed to be flipped to convert 'a' to 'b'.

Example :

```
Input : a = 10, b = 20
Output : 4
Binary representation of a is 00001010
Binary representation of b is 00010100
We need to flip highlighted four bits in a
to make it b.

Input : a = 7, b = 10
Output : 3
Binary representation of a is 00000111
Binary representation of b is 00001010
We need to flip highlighted three bits in a
to make it b.
```

1. Calculate XOR of A and B.
   a_xor_b = A ^ B
2. Count the set bits in the above
   calculated XOR result.
   countSetBits(a_xor_b)

XOR of two number will have set bits only at those places where A differs from B.

C++

```
 // Count number of bits to be flipped
// to convert A into B
#include <iostream>
using namespace std;

// Function that count set bits
int countSetBits(int n)
{
    int count = 0;
    while (n)
    {
        count += n & 1;
        n >>= 1;
    }
    return count;
}

// Function that return count of
// flipped number
int FlippedCount(int a, int b)
{
    // Return count of set bits in
    // a XOR b
    return countSetBits(a^b);
}

// Driver code
int main()
{
    int a = 10;
    int b = 20;
    cout << FlippedCount(a, b)<<endl;
    return 0;
}
```

Java

```
 // Count number of bits to be flipped
// to convert A into B
import java.util.*;

class Count {

    // Function that count set bits
    public static int countSetBits(int n)
```

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

    // Function that return count of
    // flipped number
    public static int FlippedCount(int a, int b)
    {
        // Return count of set bits in
        // a XOR b
        return countSetBits(a ^ b);
    }

    // Driver code
    public static void main(String[] args)
    {
        int a = 10;
        int b = 20;
        System.out.print(FlippedCount(a, b));
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
 # Count number of bits to be flipped
# to convert A into B

# Function that count set bits
def countSetBits( n ):
    count = 0
    while n:
        count += n & 1
        n >>= 1
    return count

# Function that return count of
# flipped number
def FlippedCount(a , b):

    # Return count of set bits in
    # a XOR b
```

```
    return countSetBits(a^b)

# Driver code
a = 10
b = 20
print(FlippedCount(a, b))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // Count number of bits to be
// flipped to convert A into B
using System;

class Count {

    // Function that count set bits
    public static int countSetBits(int n)
    {
        int count = 0;
        while (n != 0) {
            count += n & 1;
            n >>= 1;
        }
        return count;
    }

    // Function that return
    // count of flipped number
    public static int FlippedCount(int a, int b)
    {
    // Return count of set
    // bits in a XOR b
        return countSetBits(a ^ b);
    }

    // Driver code
    public static void Main()
    {
        int a = 10;
        int b = 20;
        Console.WriteLine(FlippedCount(a, b));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// Count number of bits to be
// flipped to convert A into B

// Function that count set bits
function countSetBits($n)
{
    $count = 0;
    while($n)
    {
        $count += $n & 1;
        $n >>= 1;
    }
    return $count;
}

// Function that return
// count of flipped number
function FlippedCount($a, $b)
{
    // Return count of set
    // bits in a XOR b
    return countSetBits($a ^ $b);
}

// Driver code
$a = 10;
$b = 20;
echo FlippedCount($a, $b);

// This code is contributed by mits
?>
```

Output :

```
4
```

Thanks to **Sahil Rajput** for providing above implementation.

To get the set bit count please see this post: Count set bits in an integer

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/count-number-of-bits-to-be-flipped-to-convert-a-to-b/

Chapter 99

Count number of distinct sum subsets within given range

Count number of distinct sum subsets within given range - GeeksforGeeks

Given a set S of N numbers and a range specified by two numbers L (Lower Bound) and R (Upper Bound). Find the number of distinct values of all possible sums of some subset of S that lie between the given range.

Examples :

```
Input : S = { 1, 2, 2, 3, 5 }, L = 1 and R = 5
Output : 5
Explanation :  Every number between
1 and 5 can be made out using some subset of S.
{1 as 1, 2 as 2, 3 as 3, 4 as 2 + 2 and 5 as 5}

Input : S = { 2, 3, 5 }, L = 1 and R = 7
Output : 4
Explanation :  Only 4 numbers between
1 and 7 can be made out, i.e. {2, 3, 5, 7}.
3 numbers which are {1, 4, 6} can't be made out in any way.
```

Prerequisites : Bitset | Bit Manipulation

Method 1(Simple) : A naive approach is to generate all possible subsets of given set, calculate their sum subset wise and push them into a hashmap. Iterate over the complete given range and count the numbers which exists in the hashmap.

Method 2(Efficient) : An efficient way to solve this problem is by using bitset of size 10^5. Update the bitset for every element X by left shifting the bitset and doing bitwise OR with previous bitset so that the bitset at the new possible sums become 1. Then by using the concept of **Prefix Sums**, precompute the required count of numbers between 1

and i for prefix[1..i] to answer each query in O(1) if there are more than query being asked simultaneously. For a query L and R, answer would be simply **prefix[R] − prefix[L − 1]**

> **For e.g.** S = { 2, 3, 5 }, L = 1 and R = 7
> Considering a bitset of size 32 for simplicity. Initially 1 is at 0_{th} position of bitset
> 00000000000000000000000000000001
> For incoming 2, left shifting the bitset by 2 and doing OR with previous bitset
> 00000000000000000000000000000101
> Similarly for 3,
> 00000000000000000000000000101101
> for 5,
> 00000000000000000000010110101101
> This final bitset contains 1 at those positions(possible sums) which can be made out using some
> subset of S. Hence between position 1 and 7, there are 4 set bits, thus the required answer.

Below is the implementation of above approach in C++ :

```
 // CPP Program to count the number
// distinct values of sum of some
// subset in a range
#include <bits/stdc++.h>

using namespace std;

// Constant size for bitset
#define SZ 100001

int countOfpossibleNumbers(int S[], int N,
                           int L, int R)
{
    // Creating a bitset of size SZ
    bitset <SZ> BS;

    // Set 0th position to 1
    BS[0] = 1;

    // Build the bitset
    for (int i = 0; i < N; i++) {

        // Left shift the bitset for each
        // element and taking bitwise OR
        // with previous bitset
        BS = BS | (BS << S[i]);
    }
```

```
    int prefix[SZ];

    // Intializing the prefix array to zero
    memset(prefix, 0, sizeof(prefix));

    // Build the prefix array
    for (int i = 1; i < SZ; i++) {
        prefix[i] = prefix[i - 1] + BS[i];
    }

    // Answer the given query
    int ans = prefix[R] - prefix[L - 1];

    return ans;
}

// Driver Code to test above functions
int main()
{
    int S[] = { 1, 2, 3, 5, 7 };
    int N = sizeof(S) / sizeof(S[0]);

    int L = 1, R = 18;

    cout << countOfpossibleNumbers(S, N, L, R);

    return 0;
}
```

Output:

```
18
```

Time Complexity : O(S*Z) where S*Z is the maximum sum for given constraints, i.e. 10^5

Source

https://www.geeksforgeeks.org/count-number-distinct-sum-subsets-within-given-range/

Chapter 100

Count number of subsets having a particular XOR value

Count number of subsets having a particular XOR value - GeeksforGeeks

Given an array arr[] of n numbers and a number K, find the number of subsets of arr[] having XOR of elements as K

Examples :

```
Input:   arr[]  = {6, 9, 4, 2}, k = 6
Output:  2
The subsets are {4, 2} and {6}

Input:   arr[]  = {1, 2, 3, 4, 5}, k = 4
Output:  4
The subsets are {1, 5}, {4}, {1, 2, 3, 4}
                and {2, 3, 5}
```

Brute Force approach O(2^n): One naive approach is to generate all the 2^n subsets and count all the subsets having XOR value K, but this approach will not be efficient for large values of n.

Dynamic Programming Approach O(n*m):
We define a number m such that m = pow(2,(log2(max(arr))+1)) – 1. This number is actually the maximum value any XOR subset will acquire. We get this number by counting bits in largest number. We create a 2D array dp[n+1][m+1], such that **dp[i][j] equals to the number of subsets having XOR value j from subsets of arr[0...i-1]**.

We fill the dp array as following:

1. We initialize all values of dp[i][j] as 0.
2. Set value of dp[0][0] = 1 since XOR of an empty set is 0.

3. Iterate over all the values of arr[i] from left to right and for each arr[i], iterate over all the possible values of XOR i.e from 0 to m (both inclusive) and fill the dp array asfollowing:
   for i = 1 to n:
   for j = 0 to m:
   dp[i][j] = dp[i-1][j] + dp[i-1][j^arr[i-1]]
   This can be explained as, if there is a subset arr[0…i-2] with XOR value j, then there also exists a subset arr[0…i-1] with XOR value j. also if there exists a subset arr[0….i-2] with XOR value j^arr[i] then clearly there exist a subset arr[0…i-1] with XOR value j, as j ^ arr[i-1] ^ arr[i-1] = j.
4. Counting the number of subsets with XOR value k: Since dp[i][j] is the number of subsets having j as XOR value from the subsets of arr[0..i-1], then the number of subsets from set arr[0..n] having XOR value as K will be dp[n][K]

C/C++

```
 // arr dynamic programming solution to finding the number
// of subsets having xor of their elements as k
#include<bits/stdc++.h>
using namespace std;

// Returns count of subsets of arr[] with XOR value equals
// to k.
int subsetXOR(int arr[], int n, int k)
{
    // Find maximum element in arr[]
    int max_ele = arr[0];
    for (int i=1; i<n; i++)
       if (arr[i] > max_ele)
           max_ele = arr[i];

    // Maximum possible XOR value
    int m = (1 << (int)(log2(max_ele) + 1) ) - 1;

    // The value of dp[i][j] is the number of subsets having
    // XOR of their elements as j from the set arr[0...i-1]
    int dp[n+1][m+1];

    // Initializing all the values of dp[i][j] as 0
    for (int i=0; i<=n; i++)
        for (int j=0; j<=m; j++)
            dp[i][j] = 0;

    // The xor of empty subset is 0
    dp[0][0] = 1;

    // Fill the dp table
    for (int i=1; i<=n; i++)
```

```
        for (int j=0; j<=m; j++)
            dp[i][j] = dp[i-1][j] + dp[i-1][j^arr[i-1]];

    //  The answer is the number of subset from set
    //  arr[0..n-1] having XOR of elements as k
    return dp[n][k];
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int k = 4;
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Count of subsets is " << subsetXOR(arr, n, k);
    return 0;
}
```

PHP

```
 <?php
// PHP arr dynamic programming
// solution to finding the number
// of subsets having xor of their
// elements as k

// Returns count of subsets of
// arr[] with XOR value equals to k.
function subsetXOR($arr, $n, $k)
{
    // Find maximum element in arr[]
    $max_ele = $arr[0];
    for ($i = 1; $i < $n; $i++)
    if ($arr[$i] > $max_ele)
        $max_ele = $arr[$i];

    // Maximum possible XOR value
    $m = (1 << (int)(log($max_ele,
                    2) + 1) ) - 1;

    // The value of dp[i][j] is the
    // number of subsets having
    // XOR of their elements as j
    // from the set arr[0...i-1]

    // Initializing all the
    // values of dp[i][j] as 0
    for ($i = 0; $i <= $n; $i++)
```

```
        for ($j = 0; $j <= $m; $j++)
            $dp[$i][$j] = 0;

    // The xor of empty subset is 0
    $dp[0][0] = 1;

    // Fill the dp table
    for ($i = 1; $i <= $n; $i++)
        for ( $j = 0; $j <= $m; $j++)
            $dp[$i][$j] = $dp[$i - 1][$j] +
                          $dp[$i - 1][$j ^
                          $arr[$i - 1]];

    // The answer is the number
    // of subset from set arr[0..n-1]
    // having XOR of elements as k
    return $dp[$n][$k];
}

// Driver Code
$arr = array (1, 2, 3, 4, 5);
$k = 4;
$n = sizeof($arr);
echo "Count of subsets is " ,
     subsetXOR($arr, $n, $k);

// This code is contributed by ajit
?>
```

Output :

```
Count of subsets is 4
```

This article is contributed by **Pranay Pandey**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : jit_t

Source

https://www.geeksforgeeks.org/count-number-of-subsets-having-a-particular-xor-value/

Chapter 101

Count numbers whose sum with x is equal to XOR with x

Count numbers whose sum with x is equal to XOR with x - GeeksforGeeks

Given a integer ‘x’, find the number of values of ‘a’ satisfying the following conditions:

1. 0 <= a <= x
2. a XOR x = a + x

Examples :

```
Input : 5
Output : 2
Explanation: For x = 5, following 2 values
             of 'a' satisfy the conditions:
             5 XOR 0 = 5+0
             5 XOR 2 = 5+2
Input : 10
Output : 4
Explanation: For x = 10, following 4 values
             of 'a' satisfy the conditions:
             10 XOR 0 = 10+0
             10 XOR 1 = 10+1
             10 XOR 4 = 10+4
             10 XOR 5 = 10+5
```

Naive Approach:
A Simple approach is to check for all values of ‘a’ between 0 and ‘x’ (both inclusive) and calculate its XOR with x and check if the condition 2 satisfies.

C++

```
 // C++ program to find count of values whose XOR
// with x is equal to the sum of value and x
// and values are smaller than equal to x
#include<bits/stdc++.h>
using namespace std;

int FindValues(int x)
{
    // Initialize result
    int count = 0;

    // Traversing through all values between
    // 0 and x both inclusive and counting
    // numbers that satisfy given property
    for (int i=0; i<=x; i++)
        if ((x+i) == (x^i))
            count++;

    return count;
}

// Driver code
int main()
{
    int x = 10;
    cout << FindValues(x);
    return 0;
}
```

Java

```
 // Java program to find count of values whose XOR
// with x is equal to the sum of value and x
// and values are smaller than equal to x

class Fib
{
    static int FindValues(int x)
    {
        // Initialize result
        int count = 0;

        // Traversing through all values between
        // 0 and x both inclusive and counting
        // numbers that satisfy given property
        for (int i=0; i<=x; i++)
            if ((x+i) == (x^i))
                count++;
```

```
        return count;
    }

    public static void main (String[] args)
    {
        int x=10;
        System.out.println(FindValues(x));
    }
}
```

C#

```
 // C# program to find count of values whose XOR
// with x is equal to the sum of value and x
// and values are smaller than equal to x
using System;

class Fib
{
    static int FindValues(int x)
    {
        // Initialize result
        int count = 0;

        // Traversing through all values between
        // 0 and x both inclusive and counting
        // numbers that satisfy given property
        for (int i = 0; i <= x; i++)
            if ((x+i) == (x^i))
                count++;

        return count;
    }

    // Driver code
    public static void Main ()
    {
        int x = 10;
        Console.Write(FindValues(x));
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
```

```
// PHP program to find count
// of values whose XOR with x
// is equal to the sum of value
// and x and values are smaller
// than equal to x

// function return the
// value of count
function FindValues($x)
{

    // Initialize result
    $count = 0;

    // Traversing through all values between
    // 0 and x both inclusive and counting
    // numbers that satisfy given property
    for ($i = 0; $i <= $x; $i++)
        if (($x + $i) == ($x ^ $i))
            $count++;

    return $count;
}

    // Driver code
    $x = 10;
    echo FindValues($x);

// This code is contributed by anuj_67.
?>
```

Output :

```
4
```

The time complexity of the above approach is O(x).

Efficient Approach:
XOR simulates binary addition without the carry over to the next digit. For the zero digits of 'a' we can either add a 1 or 0 without getting a carry which implies xor = + whereas if a digit in 'a' is 1 then the matching digit in x is forced to be 0 in order to avoid carry. For each 0 in 'a' in the matching digit in x can either being a 1 or 0 with a total combination count of 2^(num of zero). Hence, we just need to count the number of 0's in binary representation of the number and answer will by 2^(number of zeroes).

C++

```
 // C++ program to count numbers whose bitwise
```

```
// XOR and sum with x are equal
#include <bits/stdc++.h>
using namespace std;

// Function to find total 0 bit in a number
unsigned int CountZeroBit(int x)
{
    unsigned int count = 0;
    while (x)
    {
       if (!(x & 1))
           count++;
       x >>= 1;
    }
    return count;
}

// Function to find Count of non-negative numbers
// less than or equal to x, whose bitwise XOR and
// SUM with x are equal.
int CountXORandSumEqual(int x)
{
    // count number of zero bit in x
    int count = CountZeroBit(x);

    // power of 2 to count
    return (1 << count);
}

// Driver code
int main()
{
   int x = 10;
   cout << CountXORandSumEqual(x);
   return 0;
}
```

Java

```
 // Java program to count
// numbers whose bitwise
// XOR and sum with x
// are equal
import java.io.*;

class GFG
{
```

```
// Function to find total
// 0 bit in a number
static int CountZeroBit(int x)
{
    int count = 0;
    while (x > 0)
    {
        if ((x & 1) == 0)
            count++;
        x >>= 1;
    }
    return count;
}

// Function to find Count
// of non-negative numbers
// less than or equal to x,
// whose bitwise XOR and
// SUM with x are equal.
static int CountXORandSumEqual(int x)
{
    // count number of
    // zero bit in x
    int count = CountZeroBit(x);

    // power of 2 to count
    return (1 << count);
}

// Driver code
public static void main (String[] args)
{

    int x = 10;
    System.out.println(CountXORandSumEqual(x));
}
}

// The code is contributed by ajit
```

C#

```
 // C# program to count
// numbers whose bitwise
// XOR and sum with x
// are equal
using System;
```

```
class GFG
{

// Function to find total
// 0 bit in a number
static int CountZeroBit(int x)
{
    int count = 0;
    while (x > 0)
    {
        if ((x & 1) == 0)
            count++;
        x >>= 1;
    }
    return count;
}

// Function to find Count
// of non-negative numbers
// less than or equal to x,
// whose bitwise XOR and
// SUM with x are equal.
static int CountXORandSumEqual(int x)
{
    // count number of
    // zero bit in x
    int count = CountZeroBit(x);

    // power of 2 to count
    return (1 << count);
}

// Driver code
static public void Main ()
{
    int x = 10;
    Console.WriteLine(CountXORandSumEqual(x));
}
}

// The code is contributed by ajit
```

PHP

```
 <?php
// PHP program to count numbers whose bitwise
// XOR and sum with x are equal
```

```
// Function to find total 0 bit in a number
function CountZeroBit($x)
{
    $count = 0;
    while ($x)
    {
    if (!($x & 1))
        $count++;
    $x >>= 1;
    }
    return $count;
}

// Function to find Count of
// non-negative numbers less
// than or equal to x, whose
// bitwise XOR and SUM with
// x are equal.
function CountXORandSumEqual($x)
{
    // count number of zero bit in x
    $count = CountZeroBit($x);

    // power of 2 to count
    return (1 << $count);
}

    // Driver code
    $x = 10;
    echo CountXORandSumEqual($x);

// This code is contributed by m_kit
?>
```

Output:

```
4
```

Time complexity of this solution is O(Log x)

Improved By : nitin mittal, vt_m, jit_t

Source

https://www.geeksforgeeks.org/count-numbers-whose-sum-x-equal-xor-x/

Chapter 102

Count of divisors having more set bits than quotient on dividing N

Count of divisors having more set bits than quotient on dividing N - GeeksforGeeks

Given a positive integer N. The task is to find the number of divisor (say **d**) which give quotient (say **q**) on integer division (i.e q = N/d) such that quotient have less or equal set bit than divisor. In other words, find number of possible value of 'd' which will produce 'q' on integer dividing 'n' (q = N/d) such that 'q' have less or equal set bits (atleast 1) than 'd'.

Examples :

```
Input : N = 5
Output : 4
for d = 1 (set bit = 1),
    q = 5/1 = 5 (set bit = 2), count = 0.
for d = 2 (set bit = 1),
    q = 5/2 = 2 (set bit = 1), count = 1.
for d = 3 (set bit = 2),
    q = 5/3 = 1 (set bit = 1), count = 2.
for d = 4 (set bit = 1),
    q = 5/4 = 1 (set bit = 1), count = 3.
for d = 5 (set bit = 2),
    q = 5/5 = 1 (set bit = 1), count = 4.

Input : N = 3
Output : 2
```

Observe, for all **d** > **n**, q have 0 set bit so there is no point to go above n.

Also, for **n/2 < d <= n**, it will return q = 1, which have 1 set bit which will always be less
than or equal to d. And, minimum possible value of d is 1. So, possible value of d is from 1 to n.
Now, observe on dividing n by d, where **1 <= d <= n**, it will give q in sorted order (decreasing).
So, with increasing d we will get decreasing q. So, we get the two sorted sequence, one for increasing d
and other for decreasing q. Now, observe initially the number of set bit of d is greater than q but after
a point number of set bit of d is less than or equal to q.
So, our problem reduce to find that point i.e. 'x' for the value of d, from **1 <= d <= n**, such that q have less than or equal set bit to d.
So, count of possible d such that q have less than or equal set bit to d will be equal to n – x + 1.

Below is the implementation of this approach :

C++

```
 // C++ Program to find number of Divisors
// which on integer division produce quotient
// having less set bit than divisor
#include <bits/stdc++.h>
using namespace std;

// Return the count of set bit.
int bit(int x)
{
    int ans = 0;

    while (x) {
        x /= 2;
        ans++;
    }

    return ans;
}

// check if q and d have same number of set bit.
bool check(int d, int x)
{
    if (bit(x / d) <= bit(d))
        return true;

    return false;
}
```

```
// Binary Search to find the point at which
// number of set in q is less than or equal to d.
int bs(int n)
{
    int l = 1, r = sqrt(n);

    // while left index is less than right index
    while (l < r) {
        // finding the middle.
        int m = (l + r) / 2;

        // check if q and d have same number of
       // set it or not.
        if (check(m, n))
            r = m;
        else
            l = m + 1;
    }

    if (!check(l, n))
        return l + 1;

    else
        return l;
}

int countDivisor(int n)
{
    return n - bs(n) + 1;
}
// Driven Program
int main()
{
    int n = 5;
    cout << countDivisor(n) << endl;

    return 0;
}
```

Java

```
 // Java Program to find number
// of Divisors which on integer
// division produce quotient
// having less set bit than divisor
import java .io.*;

class GFG
```

```
{

// Return the count of set bit.
static int bit(int x)
{
    int ans = 0;
    while (x > 0)
    {
        x /= 2;
        ans++;
    }

    return ans;
}

// check if q and d have
// same number of set bit.
static boolean check(int d, int x)
{
    if (bit(x / d) <= bit(d))
        return true;

    return false;
}

// Binary Search to find
// the point at which
// number of set in q is
// less than or equal to d.
static int bs(int n)
{
    int l = 1, r = (int)Math.sqrt(n);

    // while left index is
    // less than right index
    while (l < r)
    {
        // finding the middle.
        int m = (l + r) / 2;

    // check if q and d have
    // same number of
    // set it or not.
        if (check(m, n))
            r = m;
        else
            l = m + 1;
    }
```

```
    if (!check(l, n))
        return l + 1;

    else
        return l;
}

static int countDivisor(int n)
{
    return n - bs(n) + 1;
}

    // Driver Code
    static public void main (String[] args)
    {
        int n = 5;
        System.out.println(countDivisor(n));
    }
}

// This code is contributed by anuj_67.
```

Python3

```
 # Python3 Program to find number
# of Divisors which on integer
# division produce quotient
# having less set bit than divisor
import math
# Return the count of set bit.
def bit(x) :
    ans = 0
    while (x) :
        x /= 2
        ans = ans + 1

    return ans

# check if q and d have
# same number of set bit.
def check(d, x) :
    if (bit(x /d) <= bit(d)) :
        return True

    return False;

# Binary Search to find
```

```
# the point at which
# number of set in q is
# less than or equal to d.
def bs(n) :
    l = 1
    r = int(math.sqrt(n))

    # while left index is less
    # than right index
    while (l < r) :
        # finding the middle.
        m = int((l + r) / 2)

        # check if q and d have
        # same number of
        # set it or not.
        if (check(m, n)) :
            r = m
        else :
            l = m + 1

    if (check(l, n) == False) :
        return math.floor(l + 1)
    else :
        return math.floor(l)

def countDivisor(n) :
    return n - bs(n) + 1

# Driver Code
n = 5
print (countDivisor(n))

# This code is contributed by
# Manish Shaw (manishshaw1)
```

C#

```
 // C# Program to find number of Divisors
// which on integer division produce quotient
// having less set bit than divisor
using System;
class GFG {

// Return the count of set bit.
static int bit(int x)
{
    int ans = 0;
```

```
    while (x > 0) {
        x /= 2;
        ans++;
    }

    return ans;
}

// check if q and d have
// same number of set bit.
static bool check(int d, int x)
{
    if (bit(x / d) <= bit(d))
        return true;

    return false;
}

// Binary Search to find
// the point at which
// number of set in q is
// less than or equal to d.
static int bs(int n)
{
    int l = 1, r = (int)Math.Sqrt(n);

    // while left index is
    // less than right index
    while (l < r) {
        // finding the middle.
        int m = (l + r) / 2;

    // check if q and d have
    // same number of
    // set it or not.
        if (check(m, n))
            r = m;
        else
            l = m + 1;
    }

    if (!check(l, n))
        return l + 1;

    else
        return l;
}
```

```
static int countDivisor(int n)
{
    return n - bs(n) + 1;
}

    // Driver Code
    static public void Main ()
    {
        int n = 5;
        Console.WriteLine(countDivisor(n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP Program to find number of Divisors
// which on integer division produce quotient
// having less set bit than divisor

// Return the count of set bit.
function bit( $x)
{
    $ans = 0;

    while ($x)
    {
        $x /= 2;
        $ans++;
    }

    return $ans;
}

// check if q and d have
// same number of set bit.
function check( $d, $x)
{
    if (bit($x /$d) <= bit($d))
        return true;

    return false;
}

// Binary Search to find
// the point at which
```

```
// number of set in q is
// less than or equal to d.
function bs(int $n)
{
    $l = 1;
    $r = sqrt($n);

    // while left index is less
    // than right index
    while ($l < $r)
    {

        // finding the middle.
        $m = ($l + $r) / 2;

        // check if q and d have
        // same number of
        // set it or not.
        if (check($m, $n))
            $r = $m;
        else
            $l = $m + 1;
    }

    if (!check($l, $n))
        return floor($l + 1);

    else
        return floor($l);
}

function countDivisor( $n)
{
    return $n - bs($n) + 1;
}

    // Driver Code
    $n = 5;
    echo countDivisor($n) ;

// This code is contributed by anuj_67.
?>
```

Output:

```
4
```

Improved By : vt_m, manishshaw1

Source

https://www.geeksforgeeks.org/count-divisors-set-bits-quotient-dividing-n/

Chapter 103

Count pairs in an array which have at least one digit common

Count pairs in an array which have at least one digit common - GeeksforGeeks

Given an array of N numbers. Find out the number of pairs i and j such that i < j and A_i and A_j have atleast one digit common (For e.g. (11, 19) have 1 digit common but (36, 48) have no digit common)

Examples:

> Input : A[] = { 10, 12, 24 }
> Output : 2
> **Explanation:** Two valid pairs are (10, 12) and (12, 24) which have atleast one digit common

Method 1 (Brute Force) A naive approach to solve this problem is just by running two nested loops and consider all possible pairs. We can check if the two numbers have atleast one common digit, by extracting every digit of first number and try to find it in the extracted digits of second number. The task would become much easier we simply convert them into strings.

Below is the naive implementation.

C++

```
 // CPP Program to count pairs in an array
// with some common digit
#include <bits/stdc++.h>

using namespace std;

// Returns true if the pair is valid,
```

```
// otherwise false
bool checkValidPair(int num1, int num2)
{
    // converting integers to strings
    string s1 = to_string(num1);
    string s2 = to_string(num2);

    // Iterate over the strings and check
    // if a character in first string is also
    // present in second string, return true
    for (int i = 0; i < s1.size(); i++)
        for (int j = 0; j < s2.size(); j++)
            if (s1[i] == s2[j])
                return true;

    // No common digit found
    return false;
}

// Returns the number of valid pairs
int countPairs(int arr[], int n)
{
    int numberOfPairs = 0;

    // Iterate over all possible pairs
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (checkValidPair(arr[i], arr[j]))
                numberOfPairs++;

    return numberOfPairs;
}

// Driver Code to test above functions
int main()
{
    int arr[] = { 10, 12, 24 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << countPairs(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java Program to count
// pairs in an array
// with some common digit
import java.io.*;
```

```
class GFG
{

    // Returns true if the pair
    // is valid, otherwise false
    static boolean checkValidPair(int num1,
                                  int num2)
    {
        // converting integers
        // to strings
        String s1 = Integer.toString(num1);
        String s2 = Integer.toString(num2);

        // Iterate over the strings
        // and check if a character
        // in first string is also
        // present in second string,
        // return true
        for (int i = 0; i < s1.length(); i++)
            for (int j = 0; j < s2.length(); j++)
                if (s1.charAt(i) == s2.charAt(j))
                    return true;

        // No common
        // digit found
        return false;
    }

    // Returns the number
    // of valid pairs
    static int countPairs(int []arr, int n)
    {
        int numberOfPairs = 0;

        // Iterate over all
        // possible pairs
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                if (checkValidPair(arr[i], arr[j]))
                    numberOfPairs++;

        return numberOfPairs;
    }

    // Driver Code
    public static void main(String args[])
    {
```

```
        int []arr = new int[]{ 10, 12, 24 };
        int n = arr.length;
        System.out.print(countPairs(arr, n));
    }
}

// This code is contributed
// by manish shaw.
```

C#

```
 // C# Program to count pairs in an array
// with some common digit
using System;

class GFG {

    // Returns true if the pair is valid,
    // otherwise false
    static bool checkValidPair(int num1, int num2)
    {
        // converting integers to strings
        string s1 = num1.ToString();
        string s2 = num2.ToString();

        // Iterate over the strings and check
        // if a character in first string is also
        // present in second string, return true
        for (int i = 0; i < s1.Length; i++)
            for (int j = 0; j < s2.Length; j++)
                if (s1[i] == s2[j])
                    return true;

        // No common digit found
        return false;
    }

    // Returns the number of valid pairs
    static int countPairs(int []arr, int n)
    {
        int numberOfPairs = 0;

        // Iterate over all possible pairs
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                if (checkValidPair(arr[i], arr[j]))
                    numberOfPairs++;
```

```
        return numberOfPairs;
    }

    // Driver Code to test above functions
    static void Main()
    {
        int []arr = new int[]{ 10, 12, 24 };
        int n = arr.Length;
        Console.WriteLine(countPairs(arr, n));
    }
}

// This code is contributed by manish shaw.
```

Output

```
2
```

Time Complexity: O(N^2) where N is the size of array.

Method 2 (Bit Masking): An efficient approach of solving this problem is creating a bit mask for every digit present in a particular number. Thus for every digit to be present in a number if we have a mask of 1111111111.

```
Digits -  0  1  2  3  4  5  6  7  8  9
          |  |  |  |  |  |  |  |  |  |
Mask   -  1  1  1  1  1  1  1  1  1  1

Here 1 denotes that the corresponding ith digit is set.
For e.g. 1235 can be represented as
Digits -         0  1  2  3  4  5  6  7  8  9
                 |  |  |  |  |  |  |  |  |  |
Mask for 1235 -  1  1  1  1  1  1  1  1  1  1
```

Now we just have to extract every digit of a number and make the corresponding bit set **(1 << i^{th} digit)** and store the whole number as mask. Careful analysis suggests that the maximum value of mask is 1023 in decimal (which contains all the digits from 0 – 9). Since the same set of digits can exists in more than one number, we need to maintain a frequency array to store the count of mask value.

> Let the frequencies of two masks i and j be $freq_i$ and $freq_j$ respectively,
> If(i AND j) return true, means i_{th} and j_{th} mask contains atleast one common set bit which in turn implies that the numbers from which these masks have been built also contain a common digit
> then,

increment the answer
ans += $freq_i$ * $freq_j$ [if i != j]
ans += ($freq_i$ * ($freq_i$ – 1)) / 2 [if j == i]

Below is the implementation of this efficient approach.

C++

```
 // CPP Program to count pairs in an array with
// some common digit
#include <bits/stdc++.h>
using namespace std;

// This function calculates the mask frequencies
// for every present in the array
void calculateMaskFrequencies(int arr[], int n,
                 unordered_map<int, int>& freq)
{
    // Iterate over the array
    for (int i = 0; i < n; i++) {

        int num = arr[i];

        // Creating an empty mask
        int mask = 0;

        // Extracting every digit of the number
        // and updating corresponding bit in the
        // mask
        while (num > 0) {
            mask = mask | (1 << (num % 10));
            num /= 10;
        }

        // Update the frequency array
        freq[mask]++;
    }
}

// Function return the number of valid pairs
int countPairs(int arr[], int n)
{
    // Store the mask frequencies
    unordered_map<int, int> freq;

    calculateMaskFrequencies(arr, n, freq);
```

```
    long long int numberOfPairs = 0;

    // Considering every possible pair of masks
    // and calculate pairs according to their
    // frequencies
    for (int i = 0; i < 1024; i++) {
        numberOfPairs += (freq[i] * (freq[i] - 1)) / 2;
        for (int j = i + 1; j < 1024; j++) {

            // if it contains a common digit
            if (i & j)
                numberOfPairs += (freq[i] * freq[j]);
        }
    }
    return numberOfPairs;
}

// Driver Code to test above functions
int main()
{
    int arr[] = { 10, 12, 24 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << countPairs(arr, n) << endl;
    return 0;
}
```

Time Complexity: O(N + 1024 * 1024), where N is the size of the array.

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/count-pairs-array-least-one-digit-common/

Chapter 104

Count pairs with Bitwise AND as ODD number

Count pairs with Bitwise AND as ODD number - GeeksforGeeks

Given an array of N integers. The task is to find the number of pairs (i, j) such that A[i] & A[j] is odd.

Examples:

> **Input:** N = 4
> A[] = { 5, 1, 3, 2 }
> **Output:** 3
> Since pair of A[] = (5, 1), (5, 3), (5, 2), (1, 3), (1, 2), (3, 2)
> 5 AND 1 = 1, 5 AND 3 = 1, 5 AND 2 = 0,
> 1 AND 3 = 1, 1 AND 2 = 0,
> 3 AND 2 = 2
> Total odd pair A(i, j) = 3
>
> **Input :** N = 6
> A[] = { 5, 9, 0, 6, 7, 3 }
> **Output :**6
> Since pair of A[] =
> (5, 9) = 1, (5, 0) = 0, (5, 6) = 4, (5, 7) = 5, (5, 3) = 1,
> (9, 0) = 0, (9, 6) = 0, (9, 7) = 1, (9, 3) = 1,
> (0, 6) = 0, (0, 7) = 0, (0, 3) = 0,
> (6, 7) = 6, (6, 3) = 2,
> (7, 3) = 3

A **naive approach** is to check for every pair and print the count of pairs.

Below is the implementation of the above approach:

C++

```
 // C++ program to count pairs
// with AND giving a odd number
#include <iostream>
using namespace std;

// Function to count number of odd pairs
int findOddPair(int A[], int N)
{
    int i, j;

    // variable for counting odd pairs
    int oddPair = 0;

    // find all pairs
    for (i = 0; i < N; i++) {
        for (j = i + 1; j < N; j++) {

            // find AND operation
            // check odd or even
            if ((A[i] & A[j]) % 2 != 0)
                oddPair++;
        }
    }
    // return number of odd pair
    return oddPair;
}
// Driver Code
int main()
{

    int a[] = { 5, 1, 3, 2 };
    int n = sizeof(a) / sizeof(a[0]);

    // calling function findOddPair
    // and print number of odd pair
    cout << findOddPair(a, n) << endl;

    return 0;
}
```

Java

```
 // Java program to count pairs
// with AND giving a odd number
class solution_1
{

// Function to count
```

```
// number of odd pairs
static int findOddPair(int A[],
                       int N)
{
    int i, j;

    // variable for counting
    // odd pairs
    int oddPair = 0;

    // find all pairs
    for (i = 0; i < N; i++)
    {
        for (j = i + 1; j < N; j++)
        {

            // find AND operation
            // check odd or even
            if ((A[i] & A[j]) % 2 != 0)
                oddPair++;
        }
    }

    // return number
    // of odd pair
    return oddPair;
}

// Driver Code
public static void main(String args[])
{
    int a[] = { 5, 1, 3, 2 };
    int n = a.length;

    // calling function findOddPair
    // and print number of odd pair
    System.out.println(findOddPair(a, n));
}
}

// This code is contributed
// by Arnab Kundu
```

C#

```
 // C# program to count pairs
// with AND giving a odd number
using System;
```

```
class GFG
{

// Function to count
// number of odd pairs
static int findOddPair(int []A,
                       int N)
{
    int i, j;

    // variable for counting
    // odd pairs
    int oddPair = 0;

    // find all pairs
    for (i = 0; i < N; i++)
    {
        for (j = i + 1; j < N; j++)
        {

            // find AND operation
            // check odd or even
            if ((A[i] & A[j]) % 2 != 0)
                oddPair++;
        }
    }

    // return number
    // of odd pair
    return oddPair;
}

// Driver Code
public static void Main()
{
    int []a = { 5, 1, 3, 2 };
    int n = a.Length;

    // calling function findOddPair
    // and print number of odd pair
    Console.WriteLine(findOddPair(a, n));
}
}

// This code is contributed
// inder_verma.
```

Python

```
 # Python program to count pairs
# with AND giving a odd number

# Function to count number
# of odd pairs
def findOddPair(A, N):

    # variable for counting odd pairs
    oddPair = 0

    # find all pairs
    for i in range(0, N - 1):
        for j in range(i + 1, N - 1):

            # find AND operation
            # check odd or even
            if ((A[i] & A[j]) % 2 != 0):
                oddPair = oddPair + 1

    # return number of odd pair
    return oddPair

# Driver Code
a = [5, 1, 3, 2]
n = len(a)

# calling function findOddPair
# and print number of odd pair
print(findOddPair(a, n))

# This code is contributed
# by Shivi_Aggarwal
```

PHP

```
 <?php
// PHP program to count pairs
// with AND giving a odd number

// Function to count
// number of odd pairs
function findOddPair(&$A, $N)
{

    // variable for counting
    // odd pairs
    $oddPair = 0;
```

```
    // find all pairs
    for ($i = 0; $i < $N; $i++)
    {
        for ($j = $i + 1; $j < $N; $j++)
        {

            // find AND operation
            // check odd or even
            if (($A[$i] & $A[$j]) % 2 != 0)
                $oddPair = $oddPair + 1;
        }
    }

    // return number of odd pair
    return $oddPair;
}

// Driver Code
$a = array(5, 1, 3, 2);
$n = sizeof($a);

// calling function findOddPair
// and print number of odd pair
echo(findOddPair($a, $n));

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output:

```
3
```

Time Complexity:O(N^2)

An **efficient solution** is to count the odd numbers. Then return **count * (count – 1)/2** because AND of two numbers can be odd only if only if a pair of both numbers are odd.

C++

```
 // C++ program to count pairs with Odd AND
#include <iostream>
using namespace std;

int findOddPair(int A[], int N)
{
```

```
    // Count total odd numbers in
    int count = 0;
    for (int i = 0; i < N; i++)
        if ((A[i] % 2 == 1))
            count++;

    // return count of even pair
    return count * (count - 1) / 2;
}

// Driver main
int main()
{
    int a[] = { 5, 1, 3, 2 };
    int n = sizeof(a) / sizeof(a[0]);

    // calling function findOddPair
    // and print number of odd pair
    cout << findOddPair(a, n) << endl;
    return 0;
}
```

Java

```
 // Java program to count
// pairs with Odd AND
class solution_1
{
static int findOddPair(int A[],
                       int N)
{
    // Count total odd numbers in
    int count = 0;
    for (int i = 0; i < N; i++)
        if ((A[i] % 2 == 1))
            count++;

    // return count of even pair
    return count * (count - 1) / 2;
}

// Driver Code
public static void main(String args[])
{
    int a[] = { 5, 1, 3, 2 };
    int n = a.length;

    // calling function findOddPair
```

```
    // and print number of odd pair
    System.out.println(findOddPair(a, n));

}
}

// This code is contributed
// by Arnab Kundu
```

Python

```
 # Python program to count
# pairs with Odd AND
def findOddPair(A, N):

    # Count total odd numbers
    count = 0;
    for i in range(0, N - 1):
        if ((A[i] % 2 == 1)):
            count = count+1

    # return count of even pair
    return count * (count - 1) / 2

# Driver Code
a = [5, 1, 3, 2]
n = len(a)

# calling function findOddPair
# and print number of odd pair
print(int(findOddPair(a, n)))

# This code is contributed
# by Shivi_Aggarwal
```

PHP

```
 <?php
// PHP program to count
// pairs with Odd AND
function findOddPair(&$A, $N)
{
    // Count total odd numbers
    $count = 0;
    for ($i = 0; $i < $N; $i++)
        if (($A[$i] % 2 == 1))
            $count++;
```

```
    // return count of even pair
    return $count * ($count - 1) / 2;
}

// Driver Code
$a = array(5, 1, 3, 2 );
$n = sizeof($a);

// calling function findOddPair
// and print number of odd pair
echo(findOddPair($a, $n));

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output:

```
3
```

Time Complexity: O(N)

Improved By : andrew1234, inderDuMCA, Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/count-pairs-with-bitwise-and-as-odd-number/

Chapter 105

Count pairs with Bitwise OR as Even number

Count pairs with Bitwise OR as Even number - GeeksforGeeks

Given an array A[] of size N. The task is to find the how many pair(i, j) exists such that A[i] OR A[j] is even.

Examples:

```
Input : N = 4
        A[] = { 5, 6, 2, 8 }
Output :3
Explanation :
Since pair of A[] = ( 5, 6 ), ( 5, 2 ), ( 5, 8 ),
( 6, 2 ), ( 6, 8 ), ( 2, 8 )
5 OR 6 = 7, 5 OR 2 = 7, 5 OR 8 = 13
6 OR 2 = 6, 6 OR 8 = 14, 2 OR 8 = 10
Total pair A( i, j ) = 6 and Even = 3

Input : N = 7
        A[] = {8, 6, 2, 7, 3, 4, 9}
Output :6
```

A **simple solution** is to check every pair.

C++

```
 // C++ program to count pairs with even OR
#include <iostream>
using namespace std;
```

```
int findEvenPair(int A[], int N)
{
    int evenPair = 0;
    for (int i = 0; i < N; i++) {
        for (int j = i + 1; j < N; j++) {

            // find OR operation
            // check odd or even
            if ((A[i] | A[j]) % 2 == 0)
                evenPair++;
        }
    }
    // return count of even pair
    return evenPair;
}
// Driver main
int main()
{
    int A[] = { 5, 6, 2, 8 };
    int N = sizeof(A) / sizeof(A[0]);
    cout << findEvenPair(A, N) << endl;
    return 0;
}
```

Java

```
 // Java program to count
// pairs with even OR
import java.io.*;

class GFG
{

static int findEvenPair(int A[],
                        int N)
{
    int evenPair = 0;
    for (int i = 0; i < N; i++)
    {
        for (int j = i + 1; j < N; j++)
        {

            // find OR operation
            // check odd or even
            if ((A[i] | A[j]) % 2 == 0)
                evenPair++;
        }
```

```
    }

    // return count of even pair
    return evenPair;
}

// Driver Code
public static void main (String[] args)
{
    int A[] = { 5, 6, 2, 8 };
    int N = A.length;
    System.out.println(findEvenPair(A, N));
}
}

// This code is contributed
// by inder_verma.
```

Python 3

```
 # Python 3 program to count pairs
# with even OR

def findEvenPair(A, N) :
    evenPair = 0

    for i in range(N) :
        for j in range(i+1, N):

            # find OR operation
            # check odd or even
            if (A[i] | A[j]) % 2 == 0 :
                evenPair += 1

    # return count of even pair
    return evenPair

# Driver Code
if __name__ == "__main__" :

    A = [ 5, 6, 2, 8]
    N = len(A)

    # function calling
    print(findEvenPair(A, N))

# This code is contributed by ANKITRAI1
```

C#

```
 // C# program to count
// pairs with even OR
using System;

class GFG
{

static int findEvenPair(int []A,
                        int N)
{
    int evenPair = 0;
    for (int i = 0; i < N; i++)
    {
        for (int j = i + 1; j < N; j++)
        {

            // find OR operation
            // check odd or even
            if ((A[i] | A[j]) % 2 == 0)
                evenPair++;
        }
    }

    // return count of even pair
    return evenPair;
}

// Driver Code
public static void Main ()
{
    int []A = { 5, 6, 2, 8 };
    int N = A.Length;
    Console.WriteLine(findEvenPair(A, N));
}
}

// This code is contributed
// by inder_verma.
```

PHP

```
 <?php
// PHP program to count
// pairs with even OR
function findEvenPair($A, $N)
```

```
{
    $evenPair = 0;
    for ($i = 0; $i < $N; $i++)
    {
        for ($j = $i + 1; $j < $N; $j++)
        {

            // find OR operation
            // check odd or even
            if (($A[$i] | $A[$j]) % 2 == 0)
                $evenPair++;
        }
    }
    // return count of even pair
    return $evenPair;
}

// Driver Code
$A = array(5, 6, 2, 8);
$N = count($A);
echo findEvenPair($A, $N);

// This code is contributed
// by inder_verma.
?>
```

Output:

```
3
```

Time Complexity: O(N^2)

A **efficient solution** is to count numbers with last bit as 0. Then return count * (count – 1)/2. Note that OR of two numbers can be even only if their last bits are 0.

C++

```
 // C++ program to count pairs with even OR
#include <iostream>
using namespace std;

int findEvenPair(int A[], int N)
{
    // Count total even numbers in
    // array.
    int count = 0;
    for (int i = 0; i < N; i++)
```

```
       if (!(A[i] & 1))
            count++;

    // return count of even pair
    return count * (count - 1) / 2;
}

// Driver main
int main()
{
    int A[] = { 5, 6, 2, 8 };
    int N = sizeof(A) / sizeof(A[0]);
    cout << findEvenPair(A, N) << endl;
    return 0;
}
```

Java

```
 // Java program to count
// pairs with even OR
import java.io.*;

class GFG
{
static int findEvenPair(int A[], int N)
{
    // Count total even numbers in
    // array.
    int count = 0;
    for (int i = 0; i < N; i++)
    if ((!((A[i] & 1) > 0)))
            count++;

    // return count of even pair
    return count * (count - 1) / 2;
}

// Driver Code
public static void main (String[] args)
{
    int A[] = { 5, 6, 2, 8 };
    int N = A.length;
    System.out.println(findEvenPair(A, N));
}
}

// This code is contributed
// by inder_verma.
```

C#

```
 // C# program to count
// pairs with even OR
using System;

class GFG
{
static int findEvenPair(int []A, int N)
{
    // Count total even numbers
    // in array.
    int count = 0;
    for (int i = 0; i < N; i++)
    if ((!((A[i] & 1) > 0)))
            count++;

    // return count of even pair
    return count * (count - 1) / 2;
}

// Driver Code
public static void Main (String[] args)
{
    int []A = { 5, 6, 2, 8 };
    int N = A.Length;
    Console.WriteLine(findEvenPair(A, N));
}
}

// This code is contributed
// by Kirti_Mangal
```

Output:

```
3
```

Time Complexity: O(N)

Improved By : inderDuMCA, Kirti_Mangal, ANKITRAI1

Source

https://www.geeksforgeeks.org/count-pairs-with-bitwise-or-as-even-number/

Chapter 106

Count pairs with Bitwise XOR as ODD number

Count pairs with Bitwise XOR as ODD number - GeeksforGeeks

Given an array of N integers, the task is to find the number of pairs (i, j) such that A[i] ^ A[j] is odd.

Examples:

```
Input : N = 5
        A[] =  { 5, 4, 7, 2, 1}
Output :6
Since pair of A[] =
( 5, 4 ) = 1( 5, 7 ) = 2( 5, 2 ) = 7( 5, 1 ) = 4
( 4, 7 ) = 3( 4, 2 ) = 6( 4, 1 ) = 5
( 7, 2 ) = 5( 7, 1 ) = 6
( 2, 1 ) = 3
Total XOR ODD pair  = 6

Input : N = 7
        A[] = { 7, 2, 8, 1, 0, 5, 11 }
Output :12
Since pair of A[] =
( 7, 2 ) = 5( 7, 8 ) = 15( 7, 1 ) = 6( 7, 0 ) = 7( 7, 5 ) = 2( 7, 11 ) = 12
( 2, 8 ) = 10( 2, 1 ) = 3( 2, 0 ) = 2( 2, 5 ) = 7( 2, 11 ) = 9
( 8, 1 ) = 9( 8, 0 ) = 8( 8, 5 ) = 13( 8, 11 ) = 3
( 1, 0 ) = 1( 1, 5 ) = 4( 1, 11 ) = 10
( 0, 5 ) = 5( 0, 11 ) = 11
( 5, 11 ) = 14
```

A **naive approach** is to check for every pair and print the count of pairs.

Below is the implementation of the above approach:

C++

```
 // C++ program to count pairs
// with XOR giving a odd number
#include <iostream>
using namespace std;

// Function to count number of odd pairs
int findOddPair(int A[], int N)
{
    int i, j;

    // variable for counting odd pairs
    int oddPair = 0;

    // find all pairs
    for (i = 0; i < N; i++) {
        for (j = i + 1; j < N; j++) {

            // find XOR operation
            // check odd or even
            if ((A[i] ^ A[j]) % 2 != 0)
                oddPair++;
        }
        cout << endl;
    }

    // return number of odd pair
    return oddPair;
}

// Driver Code
int main()
{

    int A[] = { 5, 4, 7, 2, 1 };
    int N = sizeof(A) / sizeof(A[0]);

    // calling function findOddPair
    // and print number of odd pair
    cout << findOddPair(A, N) << endl;

    return 0;
}
```

Java

```
 // Java program to count pairs
// with XOR giving a odd number

class GFG
{

// Function to count
// number of odd pairs
static int findOddPair(int A[],
                       int N)
{
    int i, j;

    // variable for counting
    // odd pairs
    int oddPair = 0;

    // find all pairs
    for (i = 0; i < N; i++)
    {
        for (j = i + 1; j < N; j++)
        {

            // find XOR operation
            // check odd or even
            if ((A[i] ^ A[j]) % 2 != 0)
                oddPair++;
        }

    }

    // return number
    // of odd pair
    return oddPair;
}

// Driver Code
public static void main(String args[])
{
    int A[] = { 5, 4, 7, 2, 1 };
    int N = A.length;

    // calling function findOddPair
    // and print number of odd pair
    System.out.println(findOddPair(A, N));
}
}
```

```
// This code is contributed
// by Kirti_Mangal
```

Python3

```
 # Python3 program to count pairs
# with XOR giving a odd number

# Function to count number of odd pairs
def findOddPair(A, N) :

    # variable for counting odd pairs
    oddPair = 0

    # find all pairs
    for i in range(0, N) :
        for j in range(i+1, N) :

            # find XOR operation
            # check odd or even
            if ((A[i] ^ A[j]) % 2 != 0):
                oddPair+=1

    # return number of odd pair
    return oddPair

# Driver Code
if __name__=='__main__':
    A = [5, 4, 7, 2, 1 ]
    N = len(A)

# calling function findOddPair
# and print number of odd pair
    print(findOddPair(A, N))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to count pairs
// with XOR giving a odd number
using System;

class GFG
{

// Function to count
```

```
// number of odd pairs
static int findOddPair(int[] A,
                    int N)
{
    int i, j;

    // variable for counting
    // odd pairs
    int oddPair = 0;

    // find all pairs
    for (i = 0; i < N; i++)
    {
        for (j = i + 1; j < N; j++)
        {

            // find XOR operation
            // check odd or even
            if ((A[i] ^ A[j]) % 2 != 0)
                oddPair++;
        }

    }

    // return number
    // of odd pair
    return oddPair;
}

// Driver Code
public static void Main()
{
    int[] A = { 5, 4, 7, 2, 1 };
    int N = A.Length;

    // calling function findOddPair
    // and print number of odd pair
    Console.WriteLine(findOddPair(A, N));
}
}

// This code is contributed
// by Akanksha Rai(Abby_akku)


# calling function findOddPair
# and print number of odd pair
print(findOddPair(a, n))
```

[/sourcecode]

OUTPUT:

```
6
```

Time Complexity: O(N^2)

An **efficient solution** is to count the even numbers. Then return count * (N – count).

C++

```
 // C++ program to count pairs
// with XOR giving a odd number
#include <iostream>
using namespace std;

// Function to count number of odd pairs
int findOddPair(int A[], int N)
{
    int i, count = 0;

    // find all pairs
    for (i = 0; i < N; i++) {
        if (A[i] % 2 == 0)
            count++;
    }

    // return number of odd pair
    return count * (N - count);
}

// Driver Code
int main()
{
    int a[] = { 5, 4, 7, 2, 1 };
    int n = sizeof(a) / sizeof(a[0]);

    // calling function findOddPair
    // and print number of odd pair
    cout << findOddPair(a, n) << endl;

    return 0;
}
```

Java

```
 // Java program to count pairs
// with XOR giving a odd number
class GFG
{
// Function to count
// number of odd pairs
static int findOddPair(int A[],
                       int N)
{
    int i, count = 0;

    // find all pairs
    for (i = 0; i < N; i++)
    {
        if (A[i] % 2 == 0)
            count++;
    }

    // return number of odd pair
    return count * (N - count);
}

// Driver Code
public static void main(String[] arg)
{
    int a[] = { 5, 4, 7, 2, 1 };
    int n = a.length ;

    // calling function findOddPair
    // and print number of odd pair
    System.out.println(findOddPair(a, n));
}
}

// This code is contributed
// by Smitha
```

Python3

```
 # Python3 program to count pairs
# with XOR giving a odd number

# Function to count number of odd pairs
def findOddPair(A, N) :

    count = 0

    # find all pairs
```

```
    for i in range(0 , N) :
        if (A[i] % 2 == 0) :
            count+=1

    # return number of odd pair
    return count * (N - count)

# Driver Code
if __name__=='__main__':
    a = [5, 4, 7, 2, 1]
    n = len(a)
    print(findOddPair(a,n))

# this code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to count pairs
// with XOR giving a odd number
using System;

class GFG
{
// Function to count
// number of odd pairs
static int findOddPair(int []A,
                       int N)
{
    int i, count = 0;

    // find all pairs
    for (i = 0; i < N; i++)
    {
        if (A[i] % 2 == 0)
            count++;
    }

    // return number of odd pair
    return count * (N - count);
}

// Driver Code
public static void Main()
{
    int []a = { 5, 4, 7, 2, 1 };
    int n = a.Length ;

    // calling function findOddPair
```

```
    // and print number of odd pair
    Console.Write(findOddPair(a, n));
}
}

// This code is contributed
// by Smitha
```

Output:

```
6
```

Time Complexity: O(N)

Improved By : Kirti_Mangal, Abby_akku, Smitha Dinesh Semwal, Rajput-Ji

Source

https://www.geeksforgeeks.org/count-pairs-with-bitwise-xor-as-odd-number/

Chapter 107

Count pairs with Odd XOR

Count pairs with Odd XOR - GeeksforGeeks

Given an array of n integers. Find out number of pairs in array whose XOR is odd.

Examples :

```
Input : arr[] = { 1, 2, 3 }
Output : 2
All pairs of array
1 ^ 2 = 3
1 ^ 3 = 2
2 ^ 3 = 1

Input : arr[] = { 1, 2, 3, 4 }
Output : 4
```

Naive Approach: We can find pairs whose XOR is odd by running two loops. If XOR of two number is odd increase count of pairs.

C++

```
 // C++ program to count pairs in array
// whose XOR is odd
#include <iostream>
using namespace std;

// A function will return number of pair
// whose XOR is odd
int countXorPair(int arr[], int n)
{
    // To store count of XOR pair
    int count = 0;
```

```
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++)

            // If XOR is odd increase count
            if ((arr[i] ^ arr[j]) % 2 == 1)
                count++;
    }

    // Return count
    return count;
}

// Driver program to test countXorPair()
int main()
{
    int arr[] = { 1, 2, 3 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << countXorPair(arr, n);

    return 0;
}
```

Java

```
 // Java program to count pairs in array whose
// XOR is odd
public class CountXor {
    // A function will return number of pair
    // whose XOR is odd
    static int countXorPair(int arr[], int n)
    {
        // To store count of XOR pair
        int count = 0;

        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++)

                // If XOR is odd increase count
                if ((arr[i] ^ arr[j]) % 2 == 1)
                    count++;
        }

        // Return count
        return count;
    }

    // Driver program to test countXorPair()
```

```
    public static void main(String[] args)
    {
        int arr[] = { 1, 2, 3 };
        System.out.println(countXorPair(arr, arr.length));
    }
}
```

Python 3

```
 # Python 3 program to count
# pairs in array whose XOR is odd

# A function will
# return number of pair
# whose XOR is odd
def countXorPair(arr, n):

    # To store count of XOR pair
    count = 0

    for i in range(n):
        for j in range(i + 1, n):

            # If XOR is odd increase count
            if ((arr[i] ^ arr[j]) % 2 == 1):
                count += 1

    # Return count
    return count

# Driver Code
if __name__ == "__main__":
    arr= [ 1, 2, 3 ]
    n = len(arr)
    print(countXorPair(arr, n))

# This code is contributed
# by ChitraNayal
```

C#

```
 // C# program to count pairs in
// array whose XOR is odd
using System;

public class CountXor {
```

```
    // A function will return number of pair
    // whose XOR is odd
    static int countXorPair(int[] arr, int n)
    {
        // To store count of XOR pair
        int count = 0;

        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++)

                // If XOR is odd increase count
                if ((arr[i] ^ arr[j]) % 2 == 1)
                    count++;
        }

        // Return count
        return count;
    }

    // Driver program to test countXorPair()
    public static void Main()
    {
        int[] arr = {1, 2, 3};
        Console.WriteLine(countXorPair(arr, arr.Length));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to count
// pairs in array
// whose XOR is odd

// A function will
// return number of pair
// whose XOR is odd
function countXorPair($arr,$n)
{

    // To store count
    // of XOR pair
    $count = 0;

    for ($i = 0; $i < $n; $i++)
    {
```

```
        for ($j = $i + 1; $j < $n; $j++)

            // If XOR is odd
            // increase count
            if (($arr[$i] ^ $arr[$j]) % 2 == 1)
                $count++;
    }

    // Return count
    return $count;
}

    // Driver Code
    $arr = array(1, 2, 3);
    $n = count($arr);
    echo countXorPair($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
2
```

Time Complexity : O(n*n)

Efficient Approach: We can observe that:

```
odd ^ odd = even
odd ^ even = odd
even ^ odd = odd
even ^ even = even
```

Therefore total pairs in array whose XOR is odd will be equal to count of odd numbers multiplied by count of even numbers.

C++

```
 // C++ program to count pairs in array
// whose XOR is odd
#include <iostream>
using namespace std;

// A function will return number of pair
```

```
// whose XOR is odd
int countXorPair(int arr[], int n)
{
    // To store count of odd and even
    // numbers
    int odd = 0, even = 0;

    for (int i = 0; i < n; i++) {
        // Increase even if number is
        // even otherwise increase odd
        if (arr[i] % 2 == 0)
            even++;
        else
            odd++;
    }

    // Return number of pairs
    return odd * even;
}

// Driver program to test countXorPair()
int main()
{
    int arr[] = { 1, 2, 3 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << countXorPair(arr, n);

    return 0;
}
```

Java

```
 // Java program to count pairs in array whose
// XOR is odd

public class CountXor {
    // A function will return number of pair
    // whose XOR is odd
    static int countXorPair(int arr[], int n)
    {
        // To store count of odd and even numbers
        int odd = 0, even = 0;

        for (int i = 0; i < n; i++) {
            // Increase even if number is
            // even otherwise increase odd
            if (arr[i] % 2 == 0)
                even++;
```

```
            else
                odd++;
        }

        // Return number of pairs
        return odd * even;
    }

    // Driver program to test countXorPair()
    public static void main(String[] args)
    {
        int arr[] = { 1, 2, 3 };
        System.out.println(countXorPair(arr, arr.length));
    }
}
```

Python 3

```
 # Python 3 program to count
# pairs in array whose XOR is odd

# A function will
# return number of pair
# whose XOR is odd
def countXorPair(arr, n):

    # To store count of
    # odd and even numbers
    odd = 0
    even = 0

    for i in range(n):

        # Increase even if number is
        # even otherwise increase odd
        if arr[i] % 2 == 0:
            even += 1
        else:
            odd += 1

    # Return number of pairs
    return odd * even

# Driver Code
if __name__ == "__main__":
    arr = [ 1, 2, 3 ]
    n = len(arr)
    print(countXorPair(arr, n))
```

```
# This code is contributed
# by ChitraNayal
```

C#

```
 // C# program to count pairs in
// array whose XOR is odd
using System;

public class CountXor {

    // A function will return number of pair
    // whose XOR is odd
    static int countXorPair(int[] arr, int n)
    {
        // To store count of odd and even numbers
        int odd = 0, even = 0;

        for (int i = 0; i < n; i++) {

            // Increase even if number is
            // even otherwise increase odd
            if (arr[i] % 2 == 0)
                even++;
            else
                odd++;
        }

        // Return number of pairs
        return odd * even;
    }

    // Driver program to test countXorPair()
    public static void Main()
    {
        int[] arr = {1, 2, 3};
        Console.WriteLine(countXorPair(arr, arr.Length));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to count pairs in array
```

```
// whose XOR is odd

// A function will return number of pair
// whose XOR is odd

function countXorPair($arr, $n)
{
    // To store count of odd
    // and even numbers
    $odd = 0;
    $even = 0;

    for ($i = 0; $i < $n; $i++)
    {
        // Increase even if number is
        // even otherwise increase odd
        if ($arr[$i] % 2 == 0)
            $even++;
        else
            $odd++;
    }

    // Return number of pairs
    return $odd * $even;
}

// Driver Code
$arr = array( 1, 2, 3 );
$n = sizeof($arr);
echo countXorPair($arr, $n);

// This code is contributed by Ajit_m
?>
```

Output :

```
2
```

Time Complexity : O(n)

Improved By : vt_m, jit_t, ChitraNayal

Source

https://www.geeksforgeeks.org/count-pairs-odd-xor/

Chapter 108

Count set bits in a range

Count set bits in a range - GeeksforGeeks

Given a non-negative number **n** and two values **l** and **r**. The problem is to count the number of set bits in the range **l** to **r** in the binary representation of **n**, i.e, to count set bits from the rightmost **lth** bit to the rightmost **rth** bit.

Constraint: 1 <= l <= r <= number of bits in the binary representation of **n**.

Examples:

```
Input : n = 42, l = 2, r = 5
Output : 2
(42)10 = (101010)2
There are '2' set bits in the range 2 to 5.

Input : n = 79, l = 1, r = 4
Output : 4
```

Approach: Following are the steps:

1. Calculate **num** = ((1 << r) – 1) ^ ((1 << (l-1)) – 1). This will produce a number **num** having **r** number of bits and bits in the range **l** to **r** are the only set bits.
2. Count number of set bits in the number **(n & num)**. Refer this post.

C++

```
 // C++ implementation to count set bits in the
// given range
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to get no of set bits in the
// binary representation of 'n'
unsigned int countSetBits(int n)
{
    unsigned int count = 0;
    while (n) {
        n &= (n - 1);
        count++;
    }
    return count;
}

// function to count set bits in the given range
unsigned int countSetBitsInGivenRange(unsigned int n,
                        unsigned int l, unsigned int r)
{
    // calculating a number 'num' having 'r' number
    // of bits and bits in the range l to r are the
    // only set bits
    int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

    // returns number of set bits in the range
    // 'l' to 'r' in 'n'
    return countSetBits(n & num);
}

// Driver program to test above
int main()
{
    unsigned int n = 42;
    unsigned int l = 2, r = 5;
    cout << countSetBitsInGivenRange(n, l, r);
    return 0;
}
```

Java

```
 // Java implementation to count set bits in the
// given range
class GFG {

    // Function to get no of set bits in the
    // binary representation of 'n'
    static int countSetBits(int n)
    {
        int count = 0;
```

```
        while (n > 0) {
            n &= (n - 1);
            count++;
        }

        return count;
    }

    // function to count set bits in the given range
    static int countSetBitsInGivenRange(int n, int l, int r)
    {

        // calculating a number 'num' having 'r' number
        // of bits and bits in the range l to r are the
        // only set bits
        int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

        // returns number of set bits in the range
        // 'l' to 'r' in 'n'
        return countSetBits(n & num);
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 42;
        int l = 2, r = 5;

        System.out.print(countSetBitsInGivenRange(n, l, r));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 implementation to count
# set bits in the given range

# Function to get no of set bits in the
# binary representation of 'n'
def countSetBits(n):
    count = 0
    while (n):
        n &= (n - 1)
        count = count + 1

    return count
```

```
# function to count set bits in
# the given range
def countSetBitsInGivenRange(n, l,  r):

    # calculating a number 'num' having
    # 'r' number of bits and bits in the
    # range l to r are the  only set bits
    num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1)

    # returns number of set bits in the range
    # 'l' to 'r' in 'n'
    return countSetBits(n & num)

# Driver program to test above
n = 42
l = 2
r = 5
ans = countSetBitsInGivenRange(n, l, r)
print (ans)

# This code is contributed by Saloni Gupta.
```

C#

```
 // C# implementation to count set bits in the
// given range
using System;

class GFG {

    // Function to get no of set bits in the
    // binary representation of 'n'
    static int countSetBits(int n)
    {
        int count = 0;

        while (n>0) {
            n &= (n - 1);
            count++;
        }

        return count;
    }

    // function to count set bits in the given range
    static int countSetBitsInGivenRange(int n,
                                        int l, int r)
```

```
    {

        // calculating a number 'num' having 'r' number
        // of bits and bits in the range l to r are the
        // only set bits
        int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

        // returns number of set bits in the range
        // 'l' to 'r' in 'n'
        return countSetBits(n & num);
    }

    //Driver code
    public static void Main()
    {
        int n = 42;
        int l = 2, r = 5;

        Console.WriteLine(countSetBitsInGivenRange(n, l, r));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP implementation to count
// set bits in the given range

// Function to get no of set bits in
// the binary representation of 'n'
function countSetBits($n)
{
    $count = 0;
    while ($n)
    {
        $n &= ($n - 1);
        $count++;
    }
    return $count;
}

// function to count set
// bits in the given range
function countSetBitsInGivenRange($n, $l, $r)
{
```

```
    // calculating a number 'num'
    // having 'r' number of bits
    // and bits in the range l to
    // r are the only set bits
    $num = ((1 << $r) - 1) ^
           ((1 << ($l - 1)) - 1);

    // returns number of
    // set bits in the range
    // 'l' to 'r' in 'n'
    return countSetBits($n & $num);
}

    // Driver Code
    $n = 42;
    $l = 2;
    $r = 5;
    echo countSetBitsInGivenRange($n, $l, $r);

// This code is contributed by Ajit.
?>
```

Output:

```
2
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/count-set-bits-range/

Chapter 109

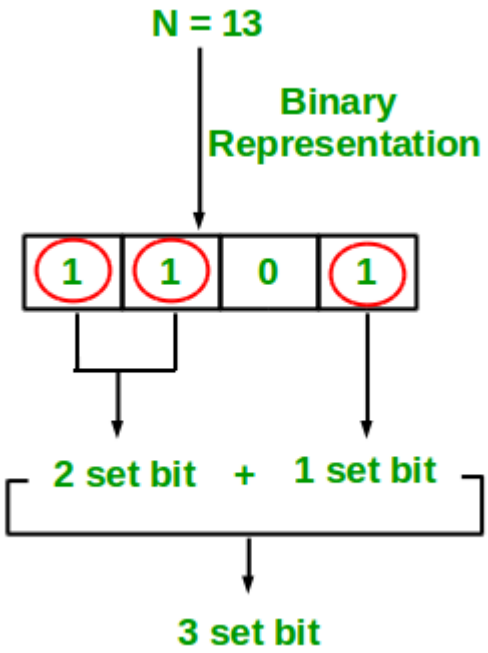
Count set bits in an integer

Count set bits in an integer - GeeksforGeeks

Write an efficient program to count number of 1s in binary representation of an integer.

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 11 is 1101 and has 3 set bits
```



1. Simple Method Loop through all bits in an integer, check if a bit is set and if it is then increment the set bit count. See below program.

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;
}

/* Program to test function countSetBits */
int main()
{
    int i = 9;
    printf("%d", countSetBits(i));
    return 0;
```

```
}
```

Java

```
 // Java program to Count set
// bits in an integer
import java.io.*;

class countSetBits
{
    /* 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;
    }

    // driver program
    public static void main(String args[])
    {
        int i = 9;
        System.out.println(countSetBits(i));
    }
}

// This code is contributed by Anshika Goyal.
```

Python3

```
 # 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 */
```

```
i = 9
print(countSetBits(i))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to Count set
// bits in an integer
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;
    }

    // Driver Code
    public static void Main()
    {
        int i = 9;
        Console.Write(countSetBits(i));
    }
}

// This code is contributed by Sam007
```

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;
}

// Driver Code
$i = 9;
echo countSetBits($i);

// This code is contributed by ajit
?>
```

Output :

```
2
```

Time Complexity: (-)(logn) (Theta of logn)

Recursive Approach :

C++

```
 // cpp implementation of recursive
// approach to find the number
// of set bits in binary representation
// of positive integer n
#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()
{
    // get value from user
    int n = 9;

    // function calling
    cout << countSetBits(n);

    return 0;
}

// This code is contributed by Raj.
```

Java

```
 // Java implementation of recursive
// approach to find the number
// of set bits in binary representation
// of positive integer n
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));
    }
}

// This code is contributes by sunnysingh
```

Python3

```
 # Python3 implementation of recursive
# approach to find the number of set
# bits in binary representation of
# positive integer n

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))

# This code is contributed by sunnysingh
```

C#

```
 // C# implementation of recursive
// approach to find the number of
// set bits in binary representation
// of positive integer n
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));
    }
}

// This code is contributed by aj_36
```

PHP

```
 <?php
// PHP implementation of recursive
// approach to find the number of
// set bits in binary representation
// of positive integer n

// 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
```

```
// get value from user
$n = 9;

// function calling
echo countSetBits($n);

// This code is contributed by m_kit.
?>
```

Output :

```
2
```

2. Brian Kernighan's Algorithm:
Subtraction of 1 from a number toggles all the bits (from right to left) till the rightmost set bit(including the rightmost set bit). So if we subtract a number by 1 and do bitwise & with itself (n & (n-1)), we unset the rightmost set bit. If we do n & (n-1) in a loop and count the no of times 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
```

Implementation of Brian Kernighan's Algorithm:

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

```
 // Java program to Count set
// bits in an integer
import java.io.*;

class countSetBits
{
    /* 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;
    }

    // driver program
    public static void main(String args[])
    {
           int i = 9;
        System.out.println(countSetBits(i));
    }
}

// This code is contributed by Anshika Goyal.
```

Python3

```
 # 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))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to Count set
// bits in an integer
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;
    }

    // Driver Code
    static public void Main ()
    {
        int i = 9;
        Console.WriteLine(countSetBits(i));
    }
```

```
}

// This code is contributed by ajit
```

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;
}

// Driver Code
$i = 9;
echo countSetBits($i);

// This code is contributed
// by akt_mit
?>
```

Output :

```
2
```

Example for Brian Kernighan's Algorithm:

```
   n =  9 (1001)
   count = 0

   Since 9 > 0, subtract by 1 and do bitwise & with (9-1)
   n = 9&8  (1001 & 1000)
   n = 8
   count  = 1
```

```
   Since 8 > 0, subtract by 1 and do bitwise & with (8-1)
   n = 8&7  (1000 & 0111)
   n = 0
   count = 2

   Since n = 0, return count which is 2 now.
```

Time Complexity: O(logn)

Recursive Approach :

C++

```
 // CPP implementation for recursive
// approach to find the number of set
// bits using Brian Kernighan's Algorithm
#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
        return 1 + countSetBits(n & (n - 1));
}

// driver code
int main()
{
    // get value from user
    int n = 9;

    // function calling
    cout << countSetBits(n);

    return 0;
}

// This code is contributed by Raj.
```

Java

```
 // Java implementation for recursive
```

```
// approach to find the number of set
// bits using Brian Kernighan Algorithm
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
            return 1 + countSetBits(n & (n - 1));
    }

    // Driver function
    public static void main(String[] args) {

        // get value from user
        int n = 9;

        // function calling
        System.out.println(countSetBits(n));
    }
}

// This code is contributed by sunnysingh
```

Python3

```
 # Python3 implementation for
# recursive approach to find
# the number of set bits using
# Brian Kernighan's Algorithm

# recursive function to count
# set bits
def countSetBits(n):

    # base case
    if (n == 0):
        return 0
    else:
        return 1 + countSetBits(n & (n - 1))


# Get value from user
```

```
n = 9

# function calling
print(countSetBits(n))

# This code is contributed by sunnysingh
```

C#

```
 // C# implementation for recursive
// approach to find the number of set
// bits using Brian Kernighan Algorithm
using System;

class GFG
{

    // recursive function
    // to count set bits
    public static int countSetBits(int n)
    {

        // base case
        if (n == 0)
            return 0;
        else
            return 1 +
                   countSetBits(n &
                                (n - 1));
    }

    // Driver Code
    static public void Main ()
    {

        // get value from user
        int n = 9;

        // function calling
        Console.WriteLine(countSetBits(n));
    }
}

// This code is contributed by aj_36
```

PHP

```
 <?php
```

```
// PHP implementation for
// recursive approach to
// find the number of set
// bits using Brian
// Kernighan's Algorithm

// recursive function to
// count set bits
function countSetBits($n)
{
    // base case
    if ($n == 0)
        return 0;
    else
        return 1 +
          countSetBits($n &
                      ($n - 1));
}

// Driver Code

// get value from user
$n = 9;

// function calling
echo countSetBits($n);

// This code is contributed by ajit.
?>
```

Output :

```
2
```

3. Using Lookup table: We can count bits in O(1) time using lookup table. Please seehttp://graphics.stanford.edu/~seander/bithacks.html#CountBitsSetTable for details.

We can find one use of counting set bits at Count number of bits to be flipped to convert A to B

Note: In GCC, we can directly count set bits using ___builtin_popcount(). So we can avoid a separate function for counting set bits.

C++

```
 // C++ program to demonstrate __builtin_popcount()
#include <iostream>
```

```
using namespace std;

int main()
{
   cout << __builtin_popcount (4) << endl;
   cout << __builtin_popcount (15);

   return 0;
}
```

PHP

```
 <?php
// PHP program to demonstrate
// __builtin_popcount()

// Driver code
$t = log10(4);
$x = log(15,2);
$tt = ceil($t);
$xx = ceil($x);

echo ($tt), "\n";
echo ($xx), "\n";

// This code is contributed
// by jit_t
?>
```

Output :

```
1
4
```

Count set bits in an integer Using Lookup Table

References:
http://graphics.stanford.edu/~seander/bithacks.html#CountBitsSetNaive

Improved By : Sam007, jit_t

Source

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

Chapter 110

Count set bits in an integer using Lookup Table

Count set bits in an integer using Lookup Table - GeeksforGeeks

Write an efficient program to count number of 1s in binary representation of an integer.

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 11 is 1101
and has 3 set bits
```

In the previous post we had seen different method that solved this problem in O(log n) time. In this post we solve in O(1) using lookup table. Here we assume that the size of INT is 32-bits. It's hard to count all 32 bits in one go using lookup table (" because it's infeasible to create lookup table of size 2^{32}-1 "). So **we break 32 bits into 8 bits of chunks(ow lookup table of size (2^8-1) index : 0-255).**

LookUp Table
In lookup tale, we store count of set_bit of every
number that are in a range (0-255)
LookupTable[0] = 0 | binary 00000000 CountSetBits 0
LookupTable[1] = 1 | binary 00000001 CountSetBits 1
LookupTable[2] = 1 | binary 00000010 CountSetBits 1
LookupTanle[3] = 2 | binary 00000011 CountSetBits 2

LookupTable[4] = 1 | binary 00000100 CountSetBits 1
and so...on upto LookupTable[255].

Let's take an Example How lookup table work.

```
Let's number be : 354
in Binary : 00000000000000000000000101100010

Split it into 8 bits chunks  :
In Binary  :  00000000 | 00000000 | 00000001 | 01100010
In decimal :     0          0          1         98

Now Count Set_bits using LookupTable
LookupTable[0] = 0
LookupTable[1] = 1
LookupTable[98] = 3

so Total bits count : 4
```

```
 // c++ count to count number of set bits
// using lookup table in O(1) time

#include <iostream>
using namespace std;

// Generate a lookup table for 32 bit integers
#define B2(n) n, n + 1, n + 1, n + 2
#define B4(n) B2(n) \, B2(n + 1), B2(n + 1), B2(n + 2)
#define B6(n) B4(n) \, B4(n + 1), B4(n + 1), B4(n + 2)

// Lookup table that store the reverse of each table
unsigned int lookuptable[256] = { B6(0), B6(1), B6(1), B6(2) };

// function countset Bits Using lookup table
// ans return set bits count
unsigned int countSetBits(int N)
{
    // first chunk of 8 bits from right
    unsigned int count = lookuptable[N & 0xff] +

                         // second chunk from  right
                         lookuptable[(N >> 8) & 0xff] +

                         // third and fourth chunks
                         lookuptable[(N >> 16) & 0xff] +
                         lookuptable[(N >> 24) & 0xff];
    return count;
```

```
}

int main()
{
    // generate lookup table
    for (int i = 0; i < 256; i++)
        lookuptable[i] = (i & 1) + lookuptable[i / 2];

    unsigned int N = 354;
    cout << countSetBits(N) << endl;

    return 0;
}
```

Output:

```
4
```

Time Complexity : O(1)

Source

https://www.geeksforgeeks.org/count-set-bits-integer-using-lookup-table/

Chapter 111

Count set bits using Python List comprehension

Count set bits using Python List comprehension - GeeksforGeeks

Write an efficient program to count number of 1s in binary representation of an integer.

Examples:

```
Input : n = 6
Output : 2
Binary representation of 6 is 110 and has 2 set bits

Input : n = 11
Output : 3
Binary representation of 11 is 1101 and has 3 set bits
```

We have existing solution for this problem please refer Count set bits in an integer link. We will solve this problem in Python using List comprehension. Approach is simple,

1. Convert given number into it's binary representation using bin(number) function.
2. Now separate out all 1's from binary representation of given number and print length of list of 1's.

```
 # Function to count set bits in an integer
# in Python

def countSetBits(num):

     # convert given number into binary
     # output will be like bin(11)=0b1101
```

```
    binary = bin(num)

    # now separate out all 1's from binary string
    # we need to skip starting two characters
    # of binary string i.e; 0b
    setBits = [ones for ones in binary[2:] if ones=='1']

    print len(setBits)

# Driver program
if __name__ == "__main__":
    num = 11
    countSetBits(num)
```

Output:

```
3
```

Source

https://www.geeksforgeeks.org/count-set-bits-using-python-list-comprehension/

Chapter 112

Count smaller numbers whose XOR with n produces greater value

Count smaller numbers whose XOR with n produces greater value - GeeksforGeeks

Given a positive integer n, count numbers x such that 0 < x <n and x^n > n where ^ is bitwise XOR operation.

Examples:

```
Input  : n = 12
Output : 3
Numbers are 1, 2 and 3
1^12 > 12,  2^12 > 12 and 3^12 > 12

Input  : n = 11
Output : 4
Numbers are 4, 5, 6 and 7
```

A number may x produce a greater XOR value if x has a set bit at a position where n has a 0 bit. So we traverse bits of n, and one by one consider all 0 bits. For every set bit at position k (Considering k = 0 for rightmost bit, k = 1 for second rightmost bit, ..), we add 2 2^k to result. For a bit at k-th position, there are 2^k numbers with set bit 1.

Below is the implementation of the above idea.

C++

```
// C++ program to count numbers whose XOR with n
```

```
// produces a value more than n.
#include<bits/stdc++.h>
using namespace std;

int countNumbers(int n)
{
    /* If there is a number like m = 11110000,
    then it is bigger then 1110xxxx. x can either
    0 or 1. So we have pow(2, k) greater numbers
    where k is  position of rightmost 1 in m. Now
    by using XOR bit at each  position can be changed.
    To change bit at any position, it needs to XOR
    it with 1. */
    int k = 0; // Position of current bit in n

    /* Traverse bits from LSB (least significant bit)
       to MSB */
    int count = 0;  // Initialize result
    while (n > 0)
    {
        // If current bit is 0, then there are
        // 2^k numbers with current bit 1 and
        // whose XOR with n produces greater value
        if ((n&1) == 0)
            count += pow(2, k);

        // Increase position for next bit
        k += 1;

        // Reduce n to find next bit
        n >>= 1;
    }

    return count;
}

// Driver code
int main()
{
    int n = 11;
    cout << countNumbers(n) << endl;
    return 0;
}
```

Java

```
 // Java program to count numbers
// whose XOR with n produces a
```

```
// value more than n.
import java.lang.*;
class GFG {

    static int countNumbers(int n)
    {

        // If there is a number like
        // m = 11110000, then it is
        // bigger than 1110xxxx. x
        // can either be 0 or 1. So
        // where k is the position of
        // rightmost 1 in m. Now by
        // using the XOR bit at each
        // position can be changed.
        // To change the bit at any
        // position, it needs to
        // XOR it with 1.
        int k = 0;
        // Position of current bit in n

        // Traverse bits from LSB (least
        // significant bit) to MSB

        int count = 0;
        // Initialize result
        while (n > 0) {
            // If the current bit is 0, then
            // there are 2^k numbers with
            // current bit 1 and whose XOR
            // with n produces greater value
            if ((n & 1) == 0)
                count += (int)(Math.pow(2, k));

            // Increase position for next bit
            k += 1;

            // Reduce n to find next bit
            n >>= 1;
        }

        return count;
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 11;
```

```
        System.out.println(countNumbers(n));
    }
}

// This code is contributed by Smitha.
```

Python3

```
 # Python program to count numbers whose
# XOR with n produces a value more than n.

def countNumbers(n):

    ''' If there is a number like m = 11110000,
    then it is bigger then 1110xxxx. x can either
    0 or 1. So we have pow(2, k) greater numbers
    where k is position of rightmost 1 in m. Now
    by using XOR bit at each position can be changed.
    To change bit at any position, it needs to XOR
    it with 1. '''

    # Position of current bit in n
    k = 0

    # Traverse bits from
    # LSB to MSB
    count = 0 # Initialize result
    while (n > 0):

        # If current bit is 0, then there are
        # 2^k numbers with current bit 1 and
        # whose XOR with n produces greater value
        if ((n & 1) == 0):
            count += pow(2, k)

        # Increase position for next bit
        k += 1

        # Reduce n to find next bit
        n >>= 1

    return count

# Driver code
n = 11
print(countNumbers(n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to count numbers
// whose XOR with n produces a
// value more than n.
using System;

class GFG {

    static int countNumbers(int n)
    {

        // If there is a number like
        // m = 11110000, then it is
        // bigger than 1110xxxx. x
        // can either be 0 or 1. So
        // where k is the position of
        // rightmost 1 in m. Now by
        // using the XOR bit at each
        // position can be changed.
        // To change the bit at any
        // position, it needs to
        // XOR it with 1.
        int k = 0;
        // Position of current bit in n

        // Traverse bits from LSB (least
        // significant bit) to MSB

        int count = 0;
        // Initialize result
        while (n > 0) {
            // If the current bit is 0, then
            // there are 2^k numbers with
            // current bit 1 and whose XOR
            // with n produces greater value
            if ((n & 1) == 0)
                count += (int)(Math.Pow(2, k));

            // Increase position for next bit
            k += 1;

            // Reduce n to find next bit
            n >>= 1;
        }

        return count;
    }
```

```
    // Driver code
    public static void Main()
    {
        int n = 11;
        Console.WriteLine(countNumbers(n));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program to count
// numbers whose XOR with n
// produces a value more than n.

function countNumbers($n)
{

    /* If there is a number
       like m = 11110000,
       then it is bigger
       then 1110xxxx. x
       can either 0 or 1.
       So we have pow(2, k)
       greater numbers where
       k is position of
       rightmost 1 in m.
       Now by using XOR bit
       at each position can
       be changed. To change
       bit at any position,
       it needs to XOR it
       with 1. */

    // Position of current bit in n
    $k = 0;

    /* Traverse bits from LSB
       (least significant bit)
       to MSB */

    // Initialize result
    $count = 0;
    while ($n > 0)
    {
```

```
        // If current bit is 0,
        // then there are 2^k
        // numbers with current
        // bit 1 and whose XOR
        // with n produces greater
        // value
        if (($n & 1) == 0)
            $count += pow(2, $k);

        // Increase position
        // for next bit
        $k += 1;

        // Reduce n to
        // find next bit
        $n >>= 1;
    }

    return $count;
}

    // Driver code
    $n = 11;
    echo countNumbers($n);

// This code is contributed by anuj_67.
?>
```

Output:

```
4
```

Time complexity : O(Log n)

Improved By : vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/count-smaller-numbers-whose-xor-n-produces-greater-value/

Chapter 113

Count smaller values whose XOR with x is greater than x

Count smaller values whose XOR with x is greater than x - GeeksforGeeks

Given a integer ‘x’, find the number of values of ‘a’ satisfying the following conditions:

1. a XOR x > x
2. 0 < a < x

Examples :

```
Input : x = 10
Output : 5
Explanation: For x = 10, following 5 values
             of 'a' satisfy the conditions:
             1 XOR 10 = 11
             4 XOR 10 = 14
             5 XOR 10 = 15
             6 XOR 10 = 12
             7 XOR 10 = 13

Input : x = 2
Output : 1
Explanation: For x=2, we have just one value
             1 XOR 2 = 3.
```

Naive Approach
A Simple approach is to check for all values of ‘a’ between 0 and ‘x’ and calculate its XOR with x

and check if the condition 1 satisfies.

C++

```
 // C++ program to find count of values
// whose XOR with x is greater than x
// and values are smaller than x
#include<bits/stdc++.h>
using namespace std;

int countValues(int x)
{
    int count = 0;
    for (int i=1; i < x; i++)
        if ((i ^ x) > x)
            count++;
    return count;
}

// Driver code
int main()
{
    int x = 10;
    cout << countValues(x);
    return 0;
}
```

Java

```
 // Java program to find count of values
// whose XOR with x is greater than x
// and values are smaller than x

public class XOR
{
    static int countValues(int x)
    {
        int count = 0;
        for (int i=1; i < x; i++)
            if ((i ^ x) > x)
                count++;
        return count;
    }

    public static void main (String[] args)
    {
        int x = 10;
```

```
        System.out.println(countValues(x));
    }
}

// This code is contributed by Saket Kumar
```

Python 3

```
 # Python3 program to find
# count of values whose
# XOR with x is greater
# than x and values are
# smaller than x

def countValues(x):

    count = 0
    for i in range(1 ,x):
        if ((i ^ x) > x):
            count += 1
    return count

# Driver code
x = 10
print(countValues(x))

# This code is contributed
# by Smitha
```

C#

```
// C# program to find count of values
// whose XOR with x is greater than x
// and values are smaller than x
using System;

class GFG
{
    static int countValues(int x)
    {
        int count = 0;
        for (int i = 1; i < x; i++)
            if ((i ^ x) > x)
                count++;
        return count;
    }
```

```
    public static void Main ()
    {
        int x = 10;
        Console.Write(countValues(x));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to find count of values
// whose XOR with x is greater than x
// and values are smaller than x

function countValues($x)
{
    $count = 0;
    for ($i = 1; $i < $x; $i++)
        if (($i ^ $x) > $x)
            $count++;
    return $count;
}

    // Driver code
    $x = 10;
    echo countValues($x);

// This code is contributed by anuj_67.
?>
```

Output :

```
5
```

The time complexity of the above approach is O(x).

Efficient Approach
Efficient solution lies in binary representation of the number. We consider all 0's in binary representation. For every 0 at i-th position, we can have 2^i numbers smaller than or equal to x with greater XOR.

C++

```
 // C++ program to find count of values
```

```
// whose XOR with x is greater than x
// and values are smaller than x
#include<bits/stdc++.h>
using namespace std;

int countValues(int x)
{
    // Initialize result
    int count = 0, n = 1;

    // Traversing through all bits of x
    while (x != 0)
    {
        // If current last bit of x is set
        // then increment count by n. Here
        // n is a power of 2 corresponding
        // to position of bit
        if (x%2 == 0)
            count += n;

        // Simultaneously calculate the 2^n
        n *= 2;

        // Replace x with x/2;
        x /= 2;
    }

    return count;
}

// Driver code
int main()
{
    int x = 10;
    cout << countValues(x);
    return 0;
}
```

Java

```
 // Java program to find count of values
// whose XOR with x is greater than x
// and values are smaller than x

class GFG
{
    static int countValues(int x)
    {
```

```
        // Initialize result
        int count = 0, n = 1;

        // Traversing through all bits of x
        while (x != 0)
        {
            // If current last bit of x is set
            // then increment count by n. Here
            // n is a power of 2 corresponding
            // to position of bit
            if (x % 2 == 0)
                count += n;

            // Simultaneously calculate the 2^n
            n *= 2;

            // Replace x with x/2;
            x /= 2;
        }
        return count;
    }

    // Driver code
    public static void main (String[] args)
    {
        int x = 10;
        System.out.println(countValues(x));
    }

}

// This code is contributed by Saket Kumar
```

Python3

```
 # Pyhton3 program to find count
# of values whose XOR with
# x is greater than x and
# values are smaller than x

def countValues(x):

    # Initialize result
    count = 0;
    n = 1;

    # Traversing through
    # all bits of x
```

```
    while (x > 0):

        # If current last bit
        # of x is set then
        # increment count by
        # n. Here n is a power
        # of 2 corresponding
        # to position of bit
        if (x % 2 == 0):
            count += n;

        # Simultaneously
        # calculate the 2^n
        n *= 2;

        # Replace x with x/2;
        x /= 2;
        x = int(x);

    return count;

# Driver code
x = 10;
print(countValues(x));

# This code is contributed
# by mits
```

C#

```
// C# program to find count of values
// whose XOR with x is greater than x
// and values are smaller than x
using System;

class GFG
{
    static int countValues(int x)
    {
        // Initialize result
        int count = 0, n = 1;

        // Traversing through all bits of x
        while (x != 0)
        {
            // If current last bit of x is set
            // then increment count by n. Here
```

```
            // n is a power of 2 corresponding
            // to position of bit
            if (x % 2 == 0)
                count += n;

            // Simultaneously calculate the 2^n
            n *= 2;

            // Replace x with x/2;
            x /= 2;
        }
        return count;
    }

    // Driver code
    public static void Main ()
    {
        int x = 10;
        Console.Write(countValues(x));
    }

}

// This code is contributed by nitin mittal
```

PHP

```
 <?php
// PHP program to find count
// of values whose XOR with
// x is greater than x and
// values are smaller than x

function countValues($x)
{

    // Initialize result
    $count = 0;
    $n = 1;

    // Traversing through
    // all bits of x
    while ($x != 0)
    {
        // If current last bit
        // of x is set then
        // increment count by
        // n. Here n is a power
```

```
        // of 2 corresponding
        // to position of bit
        if ($x % 2 == 0)
            $count += $n;

        // Simultaneously
        // calculate the 2^n
        $n *= 2;

        // Replace x with x/2;
        $x /= 2;
        $x = (int)$x;
    }

    return $count;
}

// Driver code
$x = 10;
echo countValues($x);

// This code is contributed
// by Smitha
?>
```

Output :

```
5
```

Time complexity of this solution is O(Log x)

Improved By : nitin mittal, vt_m, Smitha Dinesh Semwal, Mithun Kumar

Source

https://www.geeksforgeeks.org/count-smaller-values-whose-xor-x-greater-x/

Chapter 114

Count strings with consecutive 1's

Count strings with consecutive 1's - GeeksforGeeks

Given a number n, count number of n length strings with consecutive 1's in them.

Examples:

```
Input  : n = 2
Output : 1
There are 4 strings of length 2, the
strings are 00, 01, 10 and 11. Only the
string 11 has consecutive 1's.

Input  : n = 3
Output : 3
There are 8 strings of length 3, the
strings are 000, 001, 010, 011, 100,
101, 110 and 111.  The strings with
consecutive 1's are 011, 110 and 111.

Input : n = 5
Output : 19
```

The reverse problem of counting strings without consecutive 1's can be solved using Dynamic Programming (See the solution here). We can use that solution and find the required count using below steps.

1) Compute number of binary strings **without** consecutive 1's using the approach discussed here. Let this count be **c**.

2) Count of all possible binary strings with consecutive 1's is 2^n where n is input length.

3) Total binary strings with consecutive 1 is 2^n – c.

Below is implementation of above steps.

C++

```
 // C++ program to count all distinct
// binary strings with two consecutive 1's
#include <iostream>
using namespace std;

// Returns count of n length binary
// strings with consecutive 1's
int countStrings(int n)
{
  // Count binary strings without consecutive 1's.
  // See the approach discussed on be
  // ( http://goo.gl/p8A3sW )
    int a[n], b[n];
    a[0] = b[0] = 1;
    for (int i = 1; i < n; i++)
    {
        a[i] = a[i-1] + b[i-1];
        b[i] = a[i-1];
    }

    // Subtract a[n-1]+b[n-1] from 2^n
    return (1<<n) - a[n-1] - b[n-1];
}

// Driver program to test above functions
int main()
{
    cout << countStrings(5) << endl;
    return 0;
}
```

Java

```
 // Java program to count all distinct
// binary strings with two consecutive 1's

class GFG {

    // Returns count of n length binary
    // strings with consecutive 1's
    static int countStrings(int n)
```

```
    {
     // Count binary strings without consecutive 1's.
     // See the approach discussed on be
     // ( http://goo.gl/p8A3sW )
        int a[] = new int[n], b[] = new int[n];
        a[0] = b[0] = 1;

        for (int i = 1; i < n; i++) {
            a[i] = a[i - 1] + b[i - 1];
            b[i] = a[i - 1];
        }

       // Subtract a[n-1]+b[n-1]
 from 2^n
        return (1 << n) - a[n - 1] - b[n - 1];
    }

// Driver program to test above functions
   public static void main(String args[])
    {
        System.out.println(countStrings(5));
    }
}

//This code is contributed by Nikita tiwari.
```

Python 3

```
 # Python 3 program to count all
# distinct binary strings with
# two consecutive 1's


# Returns count of n length
# binary strings with
# consecutive 1's
def countStrings(n) :

    # Count binary strings without
    # consecutive 1's.
    # See the approach discussed on be
    # ( http://goo.gl/p8A3sW )
    a = [0] * n
    b = [0] * n
    a[0] = b[0] = 1
    for i in range(1, n) :
        a[i] = a[i - 1] + b[i - 1]
        b[i] = a[i - 1]
```

```
    # Subtract a[n-1]+b[n-1] from 2^n
    return (1 << n) - a[n - 1] - b[n - 1]

# Driver program
print(countStrings(5))


# This code is contributed
# by Nikita tiwari.
```

C#

```
 // program to count all distinct
// binary strings with two
// consecutive 1's
using System;

class GFG {

    // Returns count of n length
    // binary strings with
    // consecutive 1's
    static int countStrings(int n)
    {
        // Count binary strings without
        // consecutive 1's.
        // See the approach discussed on
        // ( http://goo.gl/p8A3sW )
        int[] a = new int[n];
        int[] b = new int[n];
        a[0] = b[0] = 1;

        for (int i = 1; i < n; i++) {
            a[i] = a[i - 1] + b[i - 1];
            b[i] = a[i - 1];
        }

        // Subtract a[n-1]+b[n-1]
        // from 2^n
        return (1 << n) - a[n - 1] - b[n - 1];
    }

    // Driver program
    public static void Main()
    {
        Console.WriteLine(countStrings(5));
```

```
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program to count all
// distinct binary strings
// with two consecutive 1's
// Returns count of n length binary
// strings with consecutive 1's

function countStrings($n)
{

    // Count binary strings without consecutive 1's.
    // See the approach discussed on be
    // ( http://goo.gl/p8A3sW )
    $a[$n] = 0;
    $b[$n] = 0;
    $a[0] = $b[0] = 1;
    for ($i = 1; $i < $n; $i++)
    {
        $a[$i] = $a[$i - 1] + $b[$i - 1];
        $b[$i] = $a[$i - 1];
    }

    // Subtract a[n-1]+b[n-1] from 2^n
    return (1 << $n) - $a[$n - 1] -
                       $b[$n - 1];
}

    // Driver Code
    echo countStrings(5), "\n";

// This Code is contributed by Ajit
?>
```

Output :

```
19
```

Optimization:
Time complexity of above solution is O(n). We can optimize above solution to work in O(Logn).

If we take a closer look at the pattern of counting strings without consecutive 1's, we can observe that the count is actually (n+2)'th Fibonacci number for n >= 1. The Fibonacci Numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 141, ….

```
n = 1, count = 0  = 21 - fib(3)
n = 2, count = 1  = 22 - fib(4)
n = 3, count = 3  = 23 - fib(5)
n = 4, count = 8  = 24 - fib(6)
n = 5, count = 19 = 24 - fib(7)
.................
```

We can find n'th Fibonacci Number in O(Log n) time (See method 4 here).

Improved By : jit_t

Source

https://www.geeksforgeeks.org/count-strings-with-consecutive-1s/

Chapter 115

Count total bits in a number

Count total bits in a number - GeeksforGeeks

Given a positive number n, count total bit in it.

Examples:

```
Input : 13
Output : 4
Binary representation of 13 is 1001

Input  : 183
Output : 8

Input  : 4096
Output : 13
```

Method 1 (Using Log)

The log2(n) logarithm in base 2 of n, which is the exponent to which 2 is raised to get n only integer and we add 1 find total bit in a number in log(n) time.

C

```
 // C program to find total bit in given number
#include <stdio.h>
#include <math.h>

unsigned countBits(unsigned int number)
{
      // log function in base 2
      // take only integer part
      return (int)log2(number)+1;
```

```
}

// Driven program
int main()
{
    unsigned int num = 65;
    printf("%d\n", countBits(num));
    return 0;
}
```

Java

```
 // Java program to
// find total bit
// in given number
import java.io.*;

class GFG
{
    static int countBits(int number)
    {

        // log function in base 2
        // take only integer part
        return (int)(Math.log(number) /
                    Math.log(2) + 1);
    }

    // Driver code
    public static void main (String[] args)
    {
        int num = 65;

        System.out.println(countBits(num));

    }
}

// This code is contributed by vij
```

Python3

```
 # Python3 program to find
# total bit in given number
import math
def countBits(number):
```

```
    # log function in base 2
    # take only integer part
    return int((math.log(number) /
                math.log(2)) + 1);

# Driver Code
num = 65;
print(countBits(num));

# This code is contributed by mits
```

C#

```
 // C# program to find total bit
// in given number
using System;

class GFG {

    static uint countBits(uint number)
    {

        // log function in base 2
        // take only integer part
        return (uint)Math.Log(number , 2.0) + 1;
    }

    // Driver code
    public static void Main()
    {
        uint num = 65;

        Console.WriteLine(countBits(num));

    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to find total
// bit in given number

function countBits($number)
{
```

```
    // log function in base 2
    // take only integer part
    return (int)(log($number) /
                   log(2)) + 1;
}

// Driver Code
$num = 65;
echo(countBits($num));

// This code is contributed by Ajit.
?>
```

Output:

```
7
```

Method 2 (Using Bit Traversal)

C

```
 /* Function to get no of bits in binary
   representation of positive integer */
#include <stdio.h>

unsigned int countBits(unsigned int n)
{
   unsigned int count = 0;
   while (n)
   {
        count++;
        n >>= 1;
    }
    return count;
}

/* Driver program*/
int main()
{
    int i = 65;
    printf("%d", countBits(i));
    return 0;
}
```

Java

```
 /* Function to get no of bits in binary
representation of positive integer */
class GFG {

    static int countBits(int n)
    {
        int count = 0;
        while (n != 0)
        {
            count++;
            n >>= 1;
        }

        return count;
    }

    /* Driver program*/
    public static void main(String[] arg)
    {
        int i = 65;
        System.out.print(countBits(i));
    }
}

// This code is contributed by Smitha.
```

Python3

```
 # Function to get no of bits
# in binary representation
# of positive integer

def countBits(n):

    count = 0
    while (n):
        count += 1
        n >>= 1

    return count

# Driver program
i = 65
print(countBits(i))

# This code is contributed
# by Smitha
```

C#

```
 /* Function to get no of bits
in binary representation of
positive integer */
using System;

class GFG
{
    static int countBits(int n)
    {
        int count = 0;
        while (n != 0)
        {
            count++;
            n >>= 1;
        }

        return count;
    }

    // Driver Code
    static public void Main ()
    {
        int i = 65;
        Console.Write(countBits(i));
    }
}

// This code is contributed
// by akt_mit.
```

PHP

```
 <?php
// PHP Code to get no of bits in binary
// representation of positive integer

// Function to get no of bits in binary
// representation of positive integer
function countBits($n)
{
    $count = 0;
    while ($n)
    {
        $count++;
        $n >>= 1;
```

```
    }
    return $count;
}

// Driver Code
$i = 65;
echo(countBits($i));

// This code is contributed by Ajit.
?>
```

Output:

```
7
```

Improved By : jit_t, Sam007, Smitha Dinesh Semwal, Mahadev99, Mithun Kumar

Source

https://www.geeksforgeeks.org/count-total-bits-number/

Chapter 116

Count total set bits in all numbers from 1 to n

Count total set bits in all numbers from 1 to n - GeeksforGeeks

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.

C++

```
 // 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);

// 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;
}
```

Java

```
 // A simple program to count set bits
// in all numbers from 1 to n.

class GFG{

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

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

        return bitCount;
    }

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

    // Driver program
    public static void main(String[] args)
    {
        int n = 4;
        System.out.print("Total set bit count is ");
        System.out.print(countSetBits(n));
    }
}

// This code is contributed by
// Smitha Dinesh Semwal
```

Python3

```
 # A simple program to count set bits
# in all numbers from 1 to n.

# Returns count of set bits present in all
# numbers from 1 to n
def countSetBits(n):

    # initialize the result
    bitCount = 0

    for i in range(1, n + 1):
        bitCount += countSetBitsUtil(i)

    return bitCount


# A utility function to count set bits
```

```
# in a number x
def countSetBitsUtil(x):

    if (x <= 0):
        return 0
    return (0 if int(x % 2) == 0 else 1) +  countSetBitsUtil(int(x / 2))


# Driver program
n = 8
print("Total set bit count is", countSetBits(n))


# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // A simple C# program to count set bits
// in all numbers from 1 to n.
using System;

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

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

        return bitCount;
    }

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

    // Driver program
```

```
    public static void Main()
    {
        int n = 4;
        Console.Write("Total set bit count is ");
        Console.Write(countSetBits(n));
    }
}

// This code is contributed by Sam007
```

Output:

```
Total set bit count is 5
```

Time Complexity: O(nLogn)

Method 2 (Simple and efficient than Method 1)
If we observe bits from rightmost side at distance i than bits get inverted after 2^i position in vertical sequence.
for example n = 5;
0 = 0000
1 = 0001
2 = 0010
3 = 0011
4 = 0100
5 = 0101

Observe the right most bit (i = 0) the bits get flipped after (2^0 = 1)
Observe the 3nd rightmost bit (i = 2) the bits get flipped after (2^2 = 4)

So, We can count bits in vertical fashion such that at i'th right most position bits will be get flipped after 2^i iteration;

C++

```
 #include <bits/stdc++.h>
using namespace std;

// Function which counts set bits from 0 to n
int countSetBits(int n)
{
    int i = 0;

    // ans store sum of set bits from 0 to n
    int ans = 0;

    // while n greater than equal to 2^i
    while ((1 << i) <= n) {
```

```
        // This k will get flipped after
        // 2^i iterations
        bool k = 0;

        // change is iterator from 2^i to 1
        int change = 1 << i;

        // This will loop from 0 to n for
        // every bit position
        for (int j = 0; j <= n; j++) {

            ans += k;

            if (change == 1) {
                k = !k; // When change = 1 flip the bit
                change = 1 << i; // again set change to 2^i
            }
            else {
                change--;
            }
        }

        // increment the position
        i++;
    }

    return ans;
}

// Main Function
int main()
{
    int n = 17;
    cout << countSetBits(n) << endl;
    return 0;
}
```

Java

```
 public class GFG {

    // Function which counts set bits from 0 to n
    static int countSetBits(int n)
    {
        int i = 0;

        // ans store sum of set bits from 0 to n
        int ans = 0;
```

```
        // while n greater than equal to 2^i
        while ((1 << i) <= n) {

            // This k will get flipped after
            // 2^i iterations
            boolean k = false;

            // change is iterator from 2^i to 1
            int change = 1 << i;

            // This will loop from 0 to n for
            // every bit position
            for (int j = 0; j <= n; j++) {

                if (k == true)
                    ans += 1;
                else
                    ans += 0;

                if (change == 1) {

                    // When change = 1 flip the bit
                    k = !k;

                    // again set change to 2^i
                    change = 1 << i;
                }
                else {
                    change--;
                }
            }

            // increment the position
            i++;
        }

        return ans;
    }

    // Driver program
    public static void main(String[] args)
    {
        int n = 17;

        System.out.println(countSetBits(n));
    }
}
```

```
// This code is contributed by Sam007.
```

C#

```
 using System;

class GFG
{
    static int countSetBits(int n)
    {
        int i = 0;

        // ans store sum of set bits from 0 to n
        int ans = 0;

        // while n greater than equal to 2^i
        while ((1 << i) <= n) {

            // This k will get flipped after
            // 2^i iterations
            bool k = false;

            // change is iterator from 2^i to 1
            int change = 1 << i;

            // This will loop from 0 to n for
            // every bit position
            for (int j = 0; j <= n; j++) {

                if (k == true)
                    ans += 1;
                else
                    ans += 0;

                if (change == 1) {

                    // When change = 1 flip the bit
                    k = !k;

                    // again set change to 2^i
                    change = 1 << i;
                }
                else {
                    change--;
                }
            }
```

```
            // increment the position
            i++;
        }

        return ans;
    }

    // Driver program
    static void Main()
    {
        int n = 17;
        Console.Write(countSetBits(n));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// Function which counts
// set bits from 0 to n
function countSetBits($n)
{
    $i = 0;

    // ans store sum of set
    // bits from 0 to n
    $ans = 0;

    // while n greater than
    // equal to 2^i
    while ((1 << $i) <= $n)
    {

        // This k will get flipped
        // after 2^i iterations
        $k = 0;

        // change is iterator
        // from 2^i to 1
        $change = 1 << $i;

        // This will loop from 0 to n
        // for every bit position
        for ($j = 0; $j <= $n; $j++)
        {
            $ans += $k;
```

```
            if ($change == 1)
            {
                // When change = 1 flip
                // the bit
                $k = !$k;

                // again set change to 2^i
                $change = 1 << $i;
            }
            else
            {
                $change--;
            }
        }

        // increment the position
        $i++;
    }

    return $ans;
}

// Driver code
$n = 17;
echo(countSetBits($n));

// This code is contributed by Smitha
?>
```

Output:

```
35
```

Time Complexity: O(k*n)
where k = number of bits to represent number n
k <= 64

Method 3 (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:

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 1 to n
#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, .. etc,
    // 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;
    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.

Here is another solution suggested by **Piyush Kapoor**.

A simple solution , using the fact that for the ith least significant bit, answer will be

```
(N/2^i)*2^(i-1)+ X
```

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;
}
```

Improved By : Ashutosh Maheshwari, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/count-total-set-bits-in-all-numbers-from-1-to-n/

Chapter 117

Count trailing zero bits using lookup table

Count trailing zero bits using lookup table - GeeksforGeeks

Given an integer, count the number of trailing zeroes. For example, for n = 12, its binary representation is 1100 and number of trailing zero bits is 2.

Examples :

```
Input : 8
Output : 3
Binary of 8 is 1000, so there are theree
trailing zero bits.

Input : 18
Output : 1
Binary of 18 is 10010, so there is one
trailing zero bit.
```

A **simple solution** is to traverse bits from LSB (Least Significant Bit) and increment count while bit is 0.

C++

```
 // Simple C++ code for counting trailing zeros
// in binary representation of a number
#include<bits/stdc++.h>
using namespace std;

int countTrailingZero(int x)
```

```
{
  int count = 0;
  while ((x & 1) == 0)
  {
      x = x >> 1;
      count++;
  }
  return count;
}

// Driver Code
int main()
{
    cout << countTrailingZero(11) << endl;
    return 0;
}
```

Java

```
 // Simple Java code for counting
// trailing zeros in binary
// representation of a number
import java.io.*;

class GFG
{
    public static int countTrailingZero(int x)
    {
        int count = 0;

        while ((x & 1) == 0)
        {
            x = x >> 1;
            count++;
        }
        return count;
    }

    // Driver Code
    public static void main (String[] args)
    {

        System.out.println(countTrailingZero(11));
    }
}

// This code is contributed by ajit
```

C#

```
 // Simple C# code for counting
// trailing zeros in binary
// representation of a number
using System;

class GFG
{
    public static int countTrailingZero(int x)
    {
        int count = 0;

        while ((x & 1) == 0)
        {
            x = x >> 1;
            count++;
        }
        return count;
    }

    // Driver Code
    static public void Main ()
    {
        Console.WriteLine(countTrailingZero(11));
    }
}

// This code is contributed by aj_36
```

PHP

```
 <?php
// Simple PHP code for counting trailing zeros
// in binary representation of a number

function countTrailingZero($x)
{
    $count = 0;
    while (($x & 1) == 0)
    {
        $x = $x >> 1;
        $count++;
    }
    return $count;
}
```

```
    // Driver Code
    echo countTrailingZero(11),"\n";

// This code is contributed by ajit
?>
```

Output :

```
0
```

Time Complexity : O(Log n)

The **lookup table solution** is based on following concepts :

1. The solution assumes that negative numbers are stored in 2's complement form which is true for most of the devices. If numbers are represented in 2's complement form, then (x & -x) [Bitwise and of x and minus x] produces a number with only last set bit.
2. Once we get a number with only one bit set, we can find its position using lookup table. It makes use of the fact that the first 32 bit position values are relatively prime with 37, so performing a modulus division with 37 gives a unique number from 0 to 36 for each. These numbers may then be mapped to the number of zeros using a small lookup table.

C++

```
 // C++ code for counting trailing zeros
// in binary representation of a number
#include<bits/stdc++.h>
using namespace std;

int countTrailingZero(int x)
{
     // Map a bit value mod 37 to its position
     static const int lookup[] = {32, 0, 1,
     26, 2, 23, 27, 0, 3, 16, 24, 30, 28, 11,
     0, 13, 4, 7, 17, 0, 25, 22, 31, 15, 29,
     10, 12, 6, 0, 21, 14, 9, 5, 20, 8, 19,
     18};

     // Only difference between (x and -x) is
     // the value of signed magnitude(leftmostbit)
     // negative numbers signed bit is 1
     return lookup[(-x & x) % 37];
}
```

```
// Driver Code
int main()
{
    cout << countTrailingZero(48) << endl;
    return 0;
}
```

Java

```
 // Java code for counting
// trailing zeros in binary
// representation of a number
import java.io.*;

class GFG
{
static int countTrailingZero(int x)
{

    // Map a bit value mod
    // 37 to its position
    int lookup[] = {32, 0, 1, 26, 2, 23,
                    27, 0, 3, 16, 24, 30,
                    28, 11, 0, 13, 4, 7,
                    17, 0, 25, 22, 31, 15,
                    29, 10, 12, 6, 0, 21,
                    14, 9, 5, 20, 8, 19, 18};

    // Only difference between
    // (x and -x) is the value
    // of signed magnitude
    // (leftmostbit) negative
    // numbers signed bit is 1
    return lookup[(-x & x) % 37];
}

// Driver Code
public static void main (String[] args)
{
    System.out.println(countTrailingZero(48));
}
}

// This code is contributed
// by ajit
```

C#

```
 // C# code for counting
// trailing zeros in binary
// representation of a number
using System;

class GFG
{
static int countTrailingZero(int x)
{

    // Map a bit value mod
    // 37 to its position
    int []lookup = {32, 0, 1, 26, 2, 23,
                    27, 0, 3, 16, 24, 30,
                    28, 11, 0, 13, 4, 7,
                    17, 0, 25, 22, 31, 15,
                    29, 10, 12, 6, 0, 21,
                    14, 9, 5, 20, 8, 19, 18};

    // Only difference between
    // (x and -x) is the value
    // of signed magnitude
    // (leftmostbit) negative
    // numbers signed bit is 1
    return lookup[(-x & x) % 37];
}

// Driver Code
static public void Main ()
{
    Console.WriteLine(countTrailingZero(48));
}
}

// This code is contributed
// by m_kit
```

PHP

```
 <?php
// PHP code for counting
// trailing zeros in binary
// representation of a number

function countTrailingZero($x)
{
    // Map a bit value mod
    // 37 to its position
```

```
    $lookup = array (32, 0, 1, 26, 2, 23,
                     27, 0, 3, 16, 24, 30,
                     28, 11, 0, 13, 4, 7,
                     17, 0, 25, 22, 31, 15,
                     29, 10, 12, 6, 0, 21,
                     14, 9, 5, 20, 8, 19, 18);

    // Only difference between
    // (x and -x) is the value
    // of signed magnitude
    // (leftmostbit) negative
    // numbers signed bit is 1
    return $lookup[(-$x &
                     $x) % 37];
}

// Driver Code
echo countTrailingZero(48), "\n";

// This code is contributed
// by akt_mit
?>
```

Output :

```
4
```

Time Complexity : O(1)

Source :
https://graphics.stanford.edu/~seander/bithacks.html

Improved By : jit_t

Source

https://www.geeksforgeeks.org/count-trailing-zero-bits-using-lookup-table/

Chapter 118

Count unset bits in a range

Count unset bits in a range - GeeksforGeeks

Given a non-negative number **n** and two values **l** and **r**. The problem is to count the number of unset bits in the range **l** to **r** in the binary representation of **n**, i.e, to count unset bits from the rightmost **lth** bit to the rightmost **rth** bit.

Examples:

```
Input : n = 42, l = 2, r = 5
Output : 2
(42)10 = (101010)2
There are '2' unset bits in the range 2 to 5.

Input : n = 80, l = 1, r = 4
Output : 4
```

Approach: Following are the steps:

1. Calculate **num** = ((1 << r) – 1) ^ ((1 << (l-1)) – 1). This will produce a number **num** having **r** number of bits and bits in the range **l** to **r** are the only set bits.
2. Count number of set bits in the number **(n & num)**. Refer this post. Let it be **count**.
3. Calculate **ans** = (r – l + 1) – count.
4. Return **ans**.

C++

```
 // C++ implementation to count unset bits in the
// given range
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to get no of set bits in the
// binary representation of 'n'
unsigned int countSetBits(int n)
{
    unsigned int count = 0;
    while (n) {
        n &= (n - 1);
        count++;
    }
    return count;
}

// function to count unset bits
// in the given range
unsigned int countUnsetBitsInGivenRange(unsigned int n,
                        unsigned int l, unsigned int r)
{
    // calculating a number 'num' having 'r' number
    // of bits and bits in the range l to r are the
    // only set bits
    int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

    // returns number of unset bits in the range
    // 'l' to 'r' in 'n'
    return (r - l + 1) - countSetBits(n & num);
}

// Driver program to test above
int main()
{
    unsigned int n = 80;
    unsigned int l = 1, r = 4;
    cout << countUnsetBitsInGivenRange(n, l, r);
    return 0;
}
```

Java

```
 // Java implementation to count unset bits in the
// given range
class GFG {

    // Function to get no of set bits in the
    // binary representation of 'n'
    static int countSetBits(int n)
    {
        int count = 0;
```

```
        while (n > 0) {
            n &= (n - 1);
            count++;
        }

        return count;
    }

    // function to count unset bits
    // in the given range
    static int countUnsetBitsInGivenRange(int n,
                                    int l, int r)
    {

        // calculating a number 'num' having 'r'
        // number of bits and bits in the range
        // l to r are the only set bits
        int num = ((1 << r) - 1) ^ ((1 <<
                                (l - 1)) - 1);

        // returns number of unset bits in the range
        // 'l' to 'r' in 'n'
        return (r - l + 1) - countSetBits(n & num);
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 80;
        int l = 1, r = 4;

        System.out.print(
            countUnsetBitsInGivenRange(n, l, r));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 implementation to count
# unset bits in the given range

# Function to get no of set bits in
# the binary representation of 'n'
def countSetBits (n):
    count = 0
```

```
    while n:
        n &= (n - 1)
        count += 1
    return count

# function to count unset bits
# in the given range
def countUnsetBitsInGivenRange (n, l, r):

    # calculating a number 'num' having
    # 'r' number of bits and bits in the
    # range l to r are the only set bits
    num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1)

    # returns number of unset bits
    # in the range 'l' to 'r' in 'n'
    return (r - l + 1) - countSetBits(n & num)

# Driver code to test above
n = 80
l = 1
r = 4
print(countUnsetBitsInGivenRange(n, l, r))

# This code is contributed by "Sharad_Bhardwaj"
```

C#

```
 // C# implementation to count unset bits in the
// given range
using System;

class GFG {

    // Function to get no of set bits in the
    // binary representation of 'n'
    static int countSetBits(int n)
    {
        int count = 0;

        while (n > 0) {
            n &= (n - 1);
            count++;
        }

        return count;
    }
```

```
    // function to count unset bits
    // in the given range
    static int countUnsetBitsInGivenRange(int n,
                                    int l,int r)
    {

        // calculating a number 'num' having 'r'
        // number of bits and bits in the range l
        // to r are the only set bits
        int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

        // returns number of unset bits in the range
        // 'l' to 'r' in 'n'
        return (r - l + 1) - countSetBits(n & num);
    }

    //Driver code
    public static void Main()
    {
        int n = 80;
        int l = 1, r = 4;

        Console.Write(countUnsetBitsInGivenRange(n, l, r));
    }
}

//This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// php implementation to count
// unset bits in the given range

// Function to get no of set bits in
// the binary representation of 'n'
function countSetBits($n)
{
    $count = 0;
    while ($n)
    {
        $n &= ($n - 1);
        $count++;
    }
    return $count;
}

// function to count unset
```

```
// bits in the given range
function countUnsetBitsInGivenRange($n, $l, $r)
{

    // calculating a number 'num'
    // having 'r' number
    // of bits and bits in the
    // range l to r are the
    // only set bits
    $num = ((1 << $r) - 1) ^
            ((1 << ($l - 1)) - 1);

    // returns number of unset
    // bits in the range
    // 'l' to 'r' in 'n'
    return ($r - $l + 1) -
           countSetBits($n & $num);
}

    // Driver code
    $n = 80;
    $l = 1;
    $r = 4;
    echo countUnsetBitsInGivenRange($n, $l, $r);

// This code is contributed by mits
?>
```

Output:

```
4
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/count-unset-bits-range/

Chapter 119

Count unset bits of a number

Count unset bits of a number - GeeksforGeeks

Given a number n, count unset bits after MSB (Most Significant Bit).

Examples :

```
Input : 17
Output : 3
Binary of 17 is 10001
so unset bit is 3

Input : 7
Output : 0
```

A **Simple Solution** is to traverse through all bits and count unset bits.

C++

```
 // C++ program to count unset bits in an integer
#include <iostream>
using namespace std;

int countunsetbits(int n)
{
    int count = 0;

    // x holds one set digit at a time
    // starting from LSB to MSB of n.
    for (int x = 1; x <= n; x = x<<1)
        if ((x & n) == 0)
            count++;
```

```
    return count;
}

// Driver code
int main()
{
    int n = 17;
    cout << countunsetbits(n);
    return 0;
}
```

Java

```
 // JAVA Code to Count unset bits in a number
class GFG {

    public static int countunsetbits(int n)
    {
        int count = 0;

        // x holds one set digit at a time
        // starting from LSB to MSB of n.
        for (int x = 1; x <= n; x = x<<1)
            if ((x & n) == 0)
                count++;

        return count;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int n = 17;
        System.out.println(countunsetbits(n));
    }
}
// This code is contributed by Arnav Kr. Mandal.
```

C#

```
 // C# Code to Count unset
// bits in a number
using System;

class GFG {
```

```
    // Function to count unset bits
    public static int countunsetbits(int n)
    {
        int count = 0;

        // x holds one set digit at a time
        // starting from LSB to MSB of n.
        for (int x = 1; x <= n; x = x << 1)
            if ((x & n) == 0)
                count++;

        return count;
    }

    // Driver Code
    public static void Main()
    {
        int n = 17;
        Console.Write(countunsetbits(n));
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHp program to count
// unset bits in an integer
function countunsetbits($n)
{
    $count = 0;

    // x holds one set digit
    // at a time starting
    // from LSB to MSB of n.
    for ($x = 1; $x <= $n;
         $x = $x << 1)
        if (($x & $n) == 0)
            $count++;

    return $count;
}

// Driver code
$n = 17;
echo countunsetbits($n);
```

```
// This code is contributed
// by nitin mittal.
?>
```

Output :

```
3
```

Above solution complexity is **log(n)**.

Efficient Solutions :
The idea is to toggle bits in O(1) time. Then apply any of the methods discussed in count set bits article.

In GCC, we can directly count set bits using ___builtin_popcount(). First toggle the bitsand then apply above function ___builtin_popcount().

C++

```
 // An optimized C++ program to count unset bits
// in an integer.
#include <iostream>
using namespace std;

int countUnsetBits(int n)
{
    int x = n;

    // Make all bits set MSB
    // (including MSB)

    // This makes sure two bits
    // (From MSB and including MSB)
    // are set
    n |= n >> 1;

    // This makes sure 4 bits
    // (From MSB and including MSB)
    // are set
    n |= n >> 2;

    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // Count set bits in toggled number
    return  __builtin_popcount(x ^ n);
}
```

```
// Driver code
int main()
{
    int n = 17;
    cout << countUnsetBits(n);
    return 0;
}
```

PHP

```
 <?php
// An optimized PHP program
// to count unset bits in
// an integer.

function countUnsetBits($n)
{
    $x = $n;

    // Make all bits set
    // MSB(including MSB)

    // This makes sure two
    // bits(From MSB and
    // including MSB) are set
    $n |= $n >> 1;

    // This makes sure 4
    // bits(From MSB and
    // including MSB) are set
    $n |= $n >> 2;

    $n |= $n >> 4;
    $n |= $n >> 8;
    $n |= $n >> 16;

    $t = log($x ^ $n,2);

    // Count set bits
    // in toggled number
    return floor($t);
}

// Driver code
$n = 17;
echo countUnsetBits($n);

// This code is contributed
```

```
// by ajit
?>
```

Output :

```
3
```

Improved By : nitin mittal, jit_t

Source

https://www.geeksforgeeks.org/count-unset-bits-number/

Chapter 120

Cyclic Redundancy Check and Modulo-2 Division

Cyclic Redundancy Check and Modulo-2 Division - GeeksforGeeks

CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in communication channel.

CRC uses **Generator Polynomial** which is available on both sender and receiver side. An example generator polynomial is of the form like $x^3 + x + 1$. This generator polynomial represents key 1011. Another example is $x^2 + 1$ that represents key 101.

```
n : Number of bits in data to be sent
    from sender side.
k : Number of bits in the key obtained
    from generator polynomial.
```

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

The binary data is first augmented by adding k-1 zeros in the end of the data

Use ***modulo-2 binary division*** to divide binary data by the key and store remainder of division.

Append the remainder at the end of the data to form the encoded data and send the same

.

- In each step, a copy of the divisor (or data) is XORed with the k bits of the dividend (or key).
- The result of the XOR operation (remainder) is (n-1) bits, which is used for the next step after 1 extra bit is pulled down to make it n bits long.

- When there are no bits left to pull down, we have a result. The (n-1)-bit remainder which is appended at the sender side.

Illustration:

Example 1 (No error in transmission):

```
Data word to be sent - 100100
Key - 1101 [ Or generator polynomial x3 + x + 1]

Sender Side:

Therefore, the remainder is 001 and hence the encoded
data sent is 100100001.

Receiver Side:
Code word received at the receiver side  100100001

Therefore, the remainder is all zeros. Hence, the
data received has no error.
```

Example 2: (Error in transmission)

```
Data word to be sent - 100100
Key - 1101

Sender Side:

Therefore, the remainder is 001 and hence the
code word sent is 100100001.

Receiver Side
Let there be error in transmission media
Code word received at the receiver side - 100000001


Since the remainder is not all zeroes, the error
is detected at the receiver side.
```

Implementation

Below is Python implementation for generating code word from given binary data and key.

```
 # Returns XOR of 'a' and 'b'
# (both of same length)
def xor(a, b):

    # initialize result
    result = []

    # Traverse all bits, if bits are
    # same, then XOR is 0, else 1
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')

    return ''.join(result)


# Performs Modulo-2 division
def mod2div(divident, divisor):

    # Number of bits to be XORed at a time.
    pick = len(divisor)

    # Slicing the divident to appropriate
    # length for particular step
    tmp = divident[0 : pick]

    while pick < len(divident):

        if tmp[0] == '1':

            # replace the divident by the result
            # of XOR and pull 1 bit down
            tmp = xor(divisor, tmp) + divident[pick]

        else:   # If leftmost bit is '0'
            # If the leftmost bit of the dividend (or the
            # part used in each step) is 0, the step cannot
            # use the regular divisor; we need to use an
            # all-0s divisor.
            tmp = xor('0'*pick, tmp) + divident[pick]

        # increment pick to move further
        pick += 1

    # For the last n bits, we have to carry it out
    # normally as increased value of pick will cause
```

```
    # Index Out of Bounds.
    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0'*pick, tmp)

    checkword = tmp
    return checkword

# Function used at the sender side to encode
# data by appending remainder of modular divison
# at the end of data.
def encodeData(data, key):

    l_key = len(key)

    # Appends n-1 zeroes at end of data
    appended_data = data + '0'*(l_key-1)
    remainder = mod2div(appended_data, key)

    # Append remainder in the original data
    codeword = data + remainder
    print("Remainder : ", remainder)
    print("Encoded Data (Data + Remainder) : ",
          codeword)

# Driver code
data = "100100"
key = "1101"
encodeData(data, key)
```

Output:

```
Remainder :  001
Encoded Data (Data + Remainder) :  100100001
```

Note that CRC is mainly designed and used to protect against common of errors on communication channels and NOT suitable protection against intentional alteration of data (See reasons here)

References:
https://en.wikipedia.org/wiki/Cyclic_redundancy_check

Source

https://www.geeksforgeeks.org/modulo-2-binary-division/

Chapter 121

Decimal Equivalent of Gray Code and its Inverse

Decimal Equivalent of Gray Code and its Inverse - GeeksforGeeks

Given a decimal number n. Find the gray code of this number in decimal form.

Examples :

> Input : 7
> Output : 4
> **Explanation:** 7 is represented as 111 in binary form. The gray code of 111 is 100 in binary form whose decimal equivalent is 4
> Input : 10
> Output : 15
> **Explanation:** 10 is represented as 1010 in binary form. The gray code of 1010 is 1111 in binary form whose decimal equivalent is 15

Following table shows the conversion of binary code values to gray code values:

Decimal Value	Binary Equivalent	Gray Code Equivalent	Decimal Value of Gray Code Equivalent
0	000	000	0
1	001	001	1
2	010	011	3
4	100	110	6
5	101	111	7
6	110	101	5
7	111	100	4

Below is the approach for the conversion of decimal code values to gray code values.

Let G(n) be Gray code equivalent of binary represention n. Consider bits of a number n and a number bit G(n). Note that leftmost set bits of both n and G(n) are at same position. Let this position be i and positions on right of it be (i+1), (i+2_, etc. The $(i + 1)^{th}$ bit in G(n) is 0 if i-th bit in n is 1 and vice versa is also true. The same is true for (i+2)-th bits, etc. Thus we have G (n) = n xor (n >> 1):

C++

```
 // CPP Program to convert given
// decimal number into decimal
// equivalent of its gray code form
#include <bits/stdc++.h>
using namespace std;

int grayCode(int n)
{
    /* Right Shift the number by 1
       taking xor with original number */
    return n ^ (n >> 1);
}

// Driver Code
int main()
{
    int n = 10;
    cout << grayCode(n) << endl;
    return 0;
}
```

Java

```
 // Java Program to convert given
// decimal number into decimal
// equivalent of its gray code form
class GFG {

    static int grayCode(int n)
    {

        // Right Shift the number
        // by 1 taking xor with
        // original number
        return n ^ (n >> 1);
    }

    // Driver Code
    public static void main(String[] args)
```

```
    {

        int n = 10;

        System.out.println(grayCode(n));
    }
}

// This code is contributed by
// Smitha Dinesh Semwal
```

Python 3

```
 # Python 3 Program to convert
# given decimal number into
# decimal equivalent of its
# gray code form

def grayCode(n):

    # Right Shift the number
    # by 1 taking xor with
    # original number
    return n ^ (n >> 1)


# Driver Code
n = 10
print(grayCode(n))

# This code is contributed
# by Smitha Dinesh Semwal
```

C#

```
 // C# Program to convert given
// decimal number into decimal
// equivalent of its gray code form
using System;

public class GFG {

    // Function for conversion
    public static int grayCode(int n)
    {

        // Right Shift the number
```

```
        // by 1 taking xor with
        // original number
        return n ^ (n >> 1);
    }

    // Driver Code
    static public void Main ()
    {
        int n = 10;

        Console.WriteLine(grayCode(n));
    }
}

// This code is contributed by Ajit.
```

PHP

```
 <?php
// PHP Program to convert given
// decimal number into decimal
// equivalent of its gray code form

function grayCode($n)
{
    /* Right Shift the number by 1
       taking xor with original
       number */
    return $n ^ ($n >> 1);
}

    // Driver Code
    $n = 10;
    echo grayCode($n) ;

// This code is contributed by nitin mittal.
?>
```

Output:

```
15
```

Finding the Gray Inverse Code

Given the decimal equivalent number n of a gray code. Find the inverse number in decimal form.

Examples:

Input : 4
Output : 7

Input : 15
Output : 10

Below is the approach for the conversion of gray code values to decimal code values.
We will go from the older bits to the younger ones (even the smallest bit has number 1, and the oldest bit is numbered k). We obtain such relations between the bits of the n_i number n and the bits of the g_i number g:

```
nk = gk,
nk-1 = gk-1 xor nk = gk xor gk-1
nk-2 = gk-2 xor nk-1 = gk xor gk-1 xor gk-2
nk-3 = gk-3 xor nk-2 = gk xor gk-1 xor gk-2 xor gk-3
...
```

C++

```
// CPP Program to convert given
// decimal number of gray code
// into its inverse in decimal form
#include <bits/stdc++.h>
using namespace std;

int inversegrayCode(int n)
{
    int inv = 0;

    // Taking xor until n becomes zero
    for (; n; n = n >> 1)
        inv ^= n;

    return inv;
}

// Driver Code
int main()
{
    int n = 15;
    cout << inversegrayCode(n) << endl;
    return 0;
}
```

Java

```
 // Java Program to convert given
// decimal number of gray code
// into its inverse in decimal form
import java.io.*;

class GFG {

    // Function to convert given
    // decimal number of gray code
    // into its inverse in decimal form
    static int inversegrayCode(int n)
    {
        int inv = 0;

        // Taking xor until n becomes zero
        for ( ; n != 0 ; n = n >> 1)
            inv ^= n;

        return inv;
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 15;
        System.out.println(inversegrayCode(n));
    }
}

// This code is contributed by Ajit.
```

Python3

```
 # Python3 Program to convert
# given decimal number of
# gray code into its inverse
# in decimal form

def inversegrayCode(n):
    inv = 0;

    # Taking xor until
    # n becomes zero
    while(n):
        inv = inv ^ n;
        n = n >> 1;
    return inv;
```

```
# Driver Code
n = 15;
print(inversegrayCode(n));

# This code is contributed
# by mits
```

C#

```
 // C# Program to convert given
// decimal number of gray code
// into its inverse in decimal form
using System;

class GFG {

    // Function to convert given
    // decimal number of gray code
    // into its inverse in decimal form
    static int inversegrayCode(int n)
    {
        int inv = 0;

        // Taking xor until n becomes zero
        for ( ; n != 0 ; n = n >> 1)
            inv ^= n;

        return inv;
    }

    // Driver code
    public static void Main ()
    {
        int n = 15;
        Console.Write(inversegrayCode(n));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP Program to convert given
// decimal number of gray code
// into its inverse in decimal form
```

```
function inversegrayCode( $n)
{
    $inv = 0;

    // Taking xor until
    // n becomes zero
    for (; $n; $n = $n >> 1)
        $inv ^= $n;

    return $inv;
}

    // Driver Code
    $n = 15;
    echo inversegrayCode($n);

// This code is contributed by anuj_67.
?>
```

Output:

```
10
```

Improved By : Smitha Dinesh Semwal, jit_t, nitin mittal, vt_m, Mithun Kumar

Source

https://www.geeksforgeeks.org/decimal-equivalent-gray-code-inverse/

Chapter 122

Decimal representation of given binary string is divisible by 10 or not

Decimal representation of given binary string is divisible by 10 or not - GeeksforGeeks

The problem is to check whether the decimal representation of the given binary number is divisible by 10 or not. Take care, the number could be very large and may not fit even in long long int. The approach should be such that there are zero or minimum number of multiplication and division operations. No leading 0's are there in the input.

Examples:

```
Input : 101000
Output : Yes
(101000)2 = (40)10
and 40 is divisible by 10.

Input : 11000111001110
Output : Yes
```

Approach: First of all we need to know that last digit of pow(2, i) = 2, 4, 8, 6 if i % 4 is equal to 1, 2, 3, 0 respectively, where i is greater than equal to 1. So, in the binary representation we need to know the position of digit '1' from the right, so as to know the perfect power of 2 with which it is going to be multiplied. This will help us to obtain the last digit of the required perfect power's of 2. We can add these digits and then check whether the last digit of the sum is 0 or not which implies that the number is divisible by 10 or not. Note that if the last digit in the binary representation is '1' then it represents an odd number, and thus not divisible by 10.

C++

```
 // C++ implementation to check whether decimal
// representation of given binary number is
// divisible by 10 or not
#include <bits/stdc++.h>
using namespace std;

// function to check whether decimal representation
// of given binary number is divisible by 10 or not
bool isDivisibleBy10(string bin)
{
    int n = bin.size();

    // if last digit is '1', then
    // number is not divisible by 10
    if (bin[n-1] == '1')
        return false;

    // to accumulate the sum of last digits
    // in perfect powers of 2
    int sum = 0;

    // traverse from the 2nd last up to 1st digit
    // in 'bin'
    for (int i=n-2; i>=0; i--)
    {
        // if digit in '1'
        if (bin[i] == '1')
        {
            // calculate digit's position from
            // the right
            int posFromRight = n - i - 1;

            // according to the digit's position,
            // obtain the last digit of the applicable
            // perfect power of 2
            if (posFromRight % 4 == 1)
                sum = sum + 2;
            else if (posFromRight % 4 == 2)
                sum = sum + 4;
            else if (posFromRight % 4 == 3)
                sum = sum + 8;
            else if (posFromRight % 4 == 0)
                sum = sum + 6;
        }
    }

    // if last digit is 0, then
    // divisible by 10
```

```
    if (sum % 10 == 0)
        return true;

    // not divisible by 10
    return false;
}

// Driver program to test above
int main()
{
    string bin = "11000111001110";

    if (isDivisibleBy10(bin))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
 // Java implementation to check whether decimal
// representation of given binary number is
// divisible by 10 or not
import java.util.*;

class GFG {

    // function to check whether decimal
    // representation of given binary number
    // is divisible by 10 or not
    static boolean isDivisibleBy10(String bin)
    {
        int n = bin.length();

        // if last digit is '1', then
        // number is not divisible by 10
        if (bin.charAt(n - 1) == '1')
            return false;

        // to accumulate the sum of last
        // digits in perfect powers of 2
        int sum = 0;

        // traverse from the 2nd last up to
        // 1st digit in 'bin'
        for (int i = n - 2; i >= 0; i--)
```

```
        {
            // if digit in '1'
            if (bin.charAt(i) == '1')
            {
                // calculate digit's position
                // from the right
                int posFromRight = n - i - 1;

                // according to the digit's
                // position, obtain the last
                // digit of the applicable
                // perfect power of 2
                if (posFromRight % 4 == 1)
                    sum = sum + 2;
                else if (posFromRight % 4 == 2)
                    sum = sum + 4;
                else if (posFromRight % 4 == 3)
                    sum = sum + 8;
                else if (posFromRight % 4 == 0)
                    sum = sum + 6;
            }
        }

        // if last digit is 0, then
        // divisible by 10
        if (sum % 10 == 0)
            return true;

        // not divisible by 10
        return false;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        String bin = "11000111001110";

        if (isDivisibleBy10(bin))
            System.out.print("Yes");
        else
            System.out.print("No");

        }
    }

// This code is contributed by Arnav Kr. Mandal.
```

Python

```
 # Python implementation to check whether
# decimal representation of given binary
# number is divisible by 10 or not


# function to check whether decimal
# representation of given binary number
# is divisible by 10 or not
def isDivisibleBy10(bin) :
    n = len(bin)

    #if last digit is '1', then
    # number is not divisible by 10
    if (bin[n - 1] == '1') :
        return False

    # to accumulate the sum of last
    # digits in perfect powers of 2
    sum = 0

    #traverse from the 2nd last up to
    # 1st digit in 'bin'

    i = n - 2
    while i >= 0 :

        # if digit in '1'
        if (bin[i] == '1') :
            # calculate digit's position
            # from the right
            posFromRight = n - i - 1

            #according to the digit's
            # position, obtain the last
            # digit of the applicable
            # perfect power of 2
            if (posFromRight % 4 == 1) :
                sum = sum + 2
            elif (posFromRight % 4 == 2) :
                sum = sum + 4
            elif (posFromRight % 4 == 3) :
                sum = sum + 8
            elif (posFromRight % 4 == 0) :
                sum = sum + 6

        i = i - 1

    # if last digit is 0, then
```

```
    # divisible by 10
    if (sum % 10 == 0) :
        return True

    # not divisible by 10
    return False


# Driver program to test above function

bin = "11000111001110"
if (isDivisibleBy10(bin)== True) :
    print("Yes")
else :
    print("No")

# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# implementation to check whether decimal
// representation of given binary number is
// divisible by 10 or not
using System;

class GFG {

    // function to check whether decimal
    // representation of given binary number
    // is divisible by 10 or not
    static bool isDivisibleBy10(String bin)
    {
        int n = bin.Length;

        // if last digit is '1', then
        // number is not divisible by 10
        if (bin[n - 1] == '1')
            return false;

        // to accumulate the sum of last
        // digits in perfect powers of 2
        int sum = 0;

        // traverse from the 2nd last up to
        // 1st digit in 'bin'
        for (int i = n - 2; i >= 0; i--) {

            // if digit in '1'
```

```
            if (bin[i] == '1') {

                // calculate digit's position
                // from the right
                int posFromRight = n - i - 1;

                // according to the digit's
                // position, obtain the last
                // digit of the applicable
                // perfect power of 2
                if (posFromRight % 4 == 1)
                    sum = sum + 2;
                else if (posFromRight % 4 == 2)
                    sum = sum + 4;
                else if (posFromRight % 4 == 3)
                    sum = sum + 8;
                else if (posFromRight % 4 == 0)
                    sum = sum + 6;
            }
        }

        // if last digit is 0, then
        // divisible by 10
        if (sum % 10 == 0)
            return true;

        // not divisible by 10
        return false;
    }

    /* Driver program to test above function */
    public static void Main()
    {
        String bin = "11000111001110";

        if (isDivisibleBy10(bin))
            Console.Write("Yes");
        else
            Console.Write("No");
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP implementation to
```

```
// check whether decimal
// representation of given
// binary number is divisible
// by 10 or not


// function to check whether
// decimal representation of
// given binary number is
// divisible by 10 or not
function isDivisibleBy10($bin)
{
    $n = strlen($bin);

    // if last digit is '1',
    // then number is not
    // divisible by 10
    if ($bin[$n - 1] == '1')
        return false;

    // to accumulate the sum
    // of last digits in
    // perfect powers of 2
    $sum = 0;

    // traverse from the 2nd
    // last up to 1st digit
    // in 'bin'
    for ($i = $n - 2; $i >= 0; $i--)
    {
        // if digit in '1'
        if ($bin[$i] == '1')
        {
            // calculate digit's
            // position from the right
            $posFromRight = $n - $i - 1;

            // according to the digit's
            // position, obtain the last
            // digit of the applicable
            // perfect power of 2
            if ($posFromRight % 4 == 1)
                $sum = $sum + 2;
            else if ($posFromRight % 4 == 2)
                $sum = $sum + 4;
            else if ($posFromRight % 4 == 3)
                $sum = $sum + 8;
            else if ($posFromRight % 4 == 0)
```

```
                $sum = $sum + 6;
            }
        }

        // if last digit is 0, then
        // divisible by 10
        if ($sum % 10 == 0)
            return true;

        // not divisible by 10
        return false;
}

// Driver Code
$bin = "11000111001110";
if(isDivisibleBy10($bin))
    echo "Yes";
else
    echo "No";

// This code is contributed by mits.
?>
```

Output:

```
Yes
```

Time Complexity: O(n), where n is the number of digits in the binary number.

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/decimal-representation-given-binary-string-divisible-10-not/

Chapter 123

Decimal representation of given binary string is divisible by 20 or not

Decimal representation of given binary string is divisible by 20 or not - GeeksforGeeks

The problem is to check whether the decimal representation of the given binary number is divisible by 20 or not. Take care, the number could be very large and may not fit even in long long int. The approach should be such that there are zero or minimum number of multiplication and division operations. No leading 0's are there in the input.

Examples :

```
Input : 101000
Output : Yes
(10100)2 = (40)10
and 40 is divisible by 20.

Input : 110001110011100
Output : Yes
```

Approach: Following are the steps:

1. Let the binary string be **bin**[].
2. Let the length of **bin**[] be **n**.
3. If bin[n-1] == '1', then it is an odd number and thus not divisible by 20.
4. Else check if bin[0..n-2] is divisible by 10. Refer this post.

C++

```
 // C++ implementation to check whether
// decimal representation of given binary
// number is divisible by 20 or not
#include <bits/stdc++.h>
using namespace std;

// function to check whether decimal
// representation of given binary number
// is divisible by 10 or not
bool isDivisibleBy10(char bin[], int n)
{
    // if last digit is '1', then
    // number is not divisible by 10
    if (bin[n - 1] == '1')
        return false;

    // to accumulate the sum of last digits
    // in perfect powers of 2
    int sum = 0;

    // traverse from the 2nd last up
    // to 1st digit in 'bin'
    for (int i = n - 2; i >= 0; i--) {

        // if digit in '1'
        if (bin[i] == '1') {

            // calculate digit's position from
            // the right
            int posFromRight = n - i - 1;

            // according to the digit's position,
            // obtain the last digit of the
            // applicable perfect power of 2
            if (posFromRight % 4 == 1)
                sum = sum + 2;
            else if (posFromRight % 4 == 2)
                sum = sum + 4;
            else if (posFromRight % 4 == 3)
                sum = sum + 8;
            else if (posFromRight % 4 == 0)
                sum = sum + 6;
        }
    }

    // if last digit is 0, then
    // divisible by 10
    if (sum % 10 == 0)
```

```
        return true;

    // not divisible by 10
    return false;
}

// function to check whether decimal
// representation of given binary number
// is divisible by 20 or not
bool isDivisibleBy20(char bin[], int n)
{
    // if 'bin' is an odd number
    if (bin[n - 1] == '1')
        return false;

    // check if bin(0..n-2) is divisible
    // by 10 or not
    return isDivisibleBy10(bin, n - 1);
}

// Driver program to test above
int main()
{
    char bin[] = "101000";
    int n = sizeof(bin) / sizeof(bin[0]);

    if (isDivisibleBy20(bin, n - 1))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
 // Java implementation to check whether
// decimal representation of given binary
// number is divisible by 20 or not
import java.io.*;

class GFG {

    // function to check whether decimal
    // representation of given binary number
    // is divisible by 10 or not
    static boolean isDivisibleBy10(char bin[], int n)
    {
```

```
    // if last digit is '1', then
    // number is not divisible by 10
    if (bin[n - 1] == '1')
        return false;

    // to accumulate the sum of last
    // digits in perfect powers of 2
    int sum = 0;

    // traverse from the 2nd last up
    // to 1st digit in 'bin'
    for (int i = n - 2; i >= 0; i--) {

        // if digit in '1'
        if (bin[i] == '1') {

            // calculate digit's position from
            // the right
            int posFromRight = n - i - 1;

            // according to the digit's position,
            // obtain the last digit of the
            // applicable perfect power of 2
            if (posFromRight % 4 == 1)
                sum = sum + 2;
            else if (posFromRight % 4 == 2)
                sum = sum + 4;
            else if (posFromRight % 4 == 3)
                sum = sum + 8;
            else if (posFromRight % 4 == 0)
                sum = sum + 6;
        }
    }

    // if last digit is 0, then
    // divisible by 10
    if (sum % 10 == 0)
        return true;

    // not divisible by 10
    return false;
}

// function to check whether decimal
// representation of given binary number
// is divisible by 20 or not
static boolean isDivisibleBy20(char bin[], int n)
{
```

```
        // if 'bin' is an odd number
        if (bin[n - 1] == '1')
            return false;

        // check if bin(0..n-2) is divisible
        // by 10 or not
        return isDivisibleBy10(bin, n - 1);
    }

    // Driver program to test above
    public static void main(String args[])
    {
        char bin[] = "101000".toCharArray();
        int n = bin.length;

        if (isDivisibleBy20(bin, n - 1))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}


// This code is contributed
// by Nikita Tiwari.
```

Python3

```
 # Python 3 implementation to check whether
# decimal representation of given binary
# number is divisible by 20 or not

# function to check whether decimal
# representation of given binary number
# is divisible by 10 or not
def isDivisibleBy10(bin, n):

    # if last digit is '1', then
    # number is not divisible by 10
    if (bin[n - 1] == '1'):
        return False

    # to accumulate the sum of last digits
    # in perfect powers of 2
    sum = 0

    # traverse from the 2nd last up
    # to 1st digit in 'bin'
```

```
    for i in range(n - 2, -1, -1):

        # if digit in '1'
        if (bin[i] == '1') :

            # calculate digit's position from
            # the right
            posFromRight = n - i - 1

            # according to the digit's position,
            # obtain the last digit of the
            # applicable perfect power of 2
            if (posFromRight % 4 == 1):
                sum = sum + 2
            elif (posFromRight % 4 == 2):
                sum = sum + 4
            elif (posFromRight % 4 == 3):
                sum = sum + 8
            elif (posFromRight % 4 == 0):
                sum = sum + 6



    # if last digit is 0, then
    # divisible by 10
    if (sum % 10 == 0):
        return True

    # not divisible by 10
    return False


# function to check whether decimal
# representation of given binary number
# is divisible by 20 or not
def isDivisibleBy20(bin, n):

    # if 'bin' is an odd number
    if (bin[n - 1] == '1'):
        return false

    # check if bin(0..n-2) is divisible
    # by 10 or not
    return isDivisibleBy10(bin, n - 1)


# Driver program to test above
bin = ['1','0','1','0','0','0']
```

```
n = len(bin)
if (isDivisibleBy20(bin, n - 1)):
    print("Yes")
else:
    print("No")

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# implementation to check whether
// decimal representation of given binary
// number is divisible by 20 or not
using System;

class GFG {

    // function to check whether decimal
    // representation of given binary number
    // is divisible by 10 or not
    static bool isDivisibleBy10(string bin, int n)
    {
        // if last digit is '1', then
        // number is not divisible by 10
        if (bin[n - 1] == '1')
            return false;

        // to accumulate the sum of last
        // digits in perfect powers of 2
        int sum = 0;

        // traverse from the 2nd last up
        // to 1st digit in 'bin'
        for (int i = n - 2; i >= 0; i--) {

            // if digit in '1'
            if (bin[i] == '1') {

                // calculate digit's position from
                // the right
                int posFromRight = n - i - 1;

                // according to the digit's position,
                // obtain the last digit of the
                // applicable perfect power of 2
                if (posFromRight % 4 == 1)
                    sum = sum + 2;
                else if (posFromRight % 4 == 2)
```

```
                sum = sum + 4;
            else if (posFromRight % 4 == 3)
                sum = sum + 8;
            else if (posFromRight % 4 == 0)
                sum = sum + 6;
        }
    }

    // if last digit is 0, then
    // divisible by 10
    if (sum % 10 == 0)
        return true;

    // not divisible by 10
    return false;
}

// function to check whether decimal
// representation of given binary number
// is divisible by 20 or not
static bool isDivisibleBy20(string bin, int n)
{
    // if 'bin' is an odd number
    if (bin[n - 1] == '1')
        return false;

    // check if bin(0..n-2) is divisible
    // by 10 or not
    return isDivisibleBy10(bin, n - 1);
}

// Driver program to test above
public static void Main()
{
    string bin = "101000";
    int n = bin.Length;

    if (isDivisibleBy20(bin, n - 1))
    Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
}
}


// This code is contributed
// by vt_m.
```

PHP

```
 <?php
// PHP implementation to check whether
// decimal representation of given binary
// number is divisible by 20 or not

// function to check whether decimal
// representation of given binary number
// is divisible by 10 or not
function isDivisibleBy10($bin, $n)
{
    // if last digit is '1', then
    // number is not divisible by 10
    if ($bin[$n - 1] == '1')
        return false;

    // to accumulate the sum of last
    // digits in perfect powers of 2
    $sum = 0;

    // traverse from the 2nd last up
    // to 1st digit in 'bin'
    for ($i = $n - 2; $i >= 0; $i--)
    {

        // if digit in '1'
        if ($bin[$i] == '1')
        {

            // calculate digit's position
            // from the right
            $posFromRight = $n - $i - 1;

            // according to the digit's position,
            // obtain the last digit of the
            // applicable perfect power of 2
            if ($posFromRight % 4 == 1)
                $sum = $sum + 2;
            else if ($posFromRight % 4 == 2)
                $sum = $sum + 4;
            else if ($posFromRight % 4 == 3)
                $sum = $sum + 8;
            else if ($posFromRight % 4 == 0)
                $sum = $sum + 6;
        }
    }
```

```
    // if last digit is 0, then
    // divisible by 10
    if ($sum % 10 == 0)
        return true;

    // not divisible by 10
    return false;
}

// function to check whether decimal
// representation of given binary number
// is divisible by 20 or not
function isDivisibleBy20($bin, $n)
{
    // if 'bin' is an odd number
    if ($bin[$n - 1] == '1')
        return false;

    // check if bin(0..n-2) is divisible
    // by 10 or not
    return isDivisibleBy10($bin, $n - 1);
}

// Driver code
$bin= "101000";
$n = strlen($bin);

if (isDivisibleBy20($bin, $n - 1))
    echo "Yes";
else
    echo "No";

// This code is contributed by mits
?>
```

Output :

```
Yes
```

Time Complexity: O(n), where n is the number of digits in the binary number.

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/decimal-representation-given-binary-string-divisible-20-not/

Chapter 124

Detect if two integers have opposite signs

Detect if two integers have opposite signs - GeeksforGeeks

Given two signed integers, write a function that returns true if the signs of given integers are different, otherwise false. For example, the function should return true -1 and +100, and should return false for -100 and -200. The function should not use any of the arithmetic operators.

Let the given integers be x and y. The sign bit is 1 in negative numbers, and 0 in positive numbers. The XOR of x and y will have the sign bit as 1 iff they have opposite sign. In other words, XOR of x and y will be negative number number iff x and y have opposite signs. The following code use this logic.

C++

```
 // C++ Program to Detect
// if two integers have opposite signs.
#include<stdbool.h>
#include<stdio.h>

bool oppositeSigns(int x, int y)
{
    return ((x ^ y) < 0);
}

int main()
{
    int x = 100, y = -100;
    if (oppositeSigns(x, y) == true)
       printf ("Signs are opposite");
    else
```

```
      printf ("Signs are not opposite");
    return 0;
}
```

Java

```
 // Java Program to Detect
// if two integers have opposite signs.

class GFG {

    static boolean oppositeSigns(int x, int y)
    {
        return ((x ^ y) < 0);
    }

    public static void main(String[] args)
    {
        int x = 100, y = -100;
        if (oppositeSigns(x, y) == true)
            System.out.println("Signs are opposite");
        else
            System.out.println("Signs are not opposite");
    }
}

// This code is contributed by prerna saini.
```

Python3

```
 # Python3 Program to Detect
# if two integers have
# opposite signs.
def oppositeSigns(x, y):
    return ((x ^ y) < 0);

x = 100
y = 1

if (oppositeSigns(x, y) == True):
    print "Signs are opposite"
else:
    print "Signs are not opposite"

# This article is contributed by Prerna Saini.
```

C#

```
 // C# Program to Detect
// if two integers have
// opposite signs.
using System;

class GFG {

    // Function to detect signs
    static bool oppositeSigns(int x, int y)
    {
        return ((x ^ y) < 0);
    }

    // Driver Code
    public static void Main()
    {
        int x = 100, y = -100;
        if (oppositeSigns(x, y) == true)
            Console.Write("Signs are opposite");
        else
            Console.Write("Signs are not opposite");
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP Program to Detect if two
// integers have opposite signs.

function oppositeSigns($x, $y)
{
    return (($x ^ $y) < 0);
}

    // Driver Code
    $x = 100;
    $y = -100;
    if (oppositeSigns($x, $y) == true)
    echo ("Signs are opposite");
    else
    echo ("Signs are not opposite");

// This code is contributed by vt_m.
?>
```

Output:

```
Signs are opposite
```

Source: Detect if two integers have opposite signs

We can also solve this by using two comparison operators. See the following code.

```
 bool oppositeSigns(int x, int y)
{
    return (x < 0)? (y >= 0): (y < 0);
}
```

The first method is more efficient. The first method uses a bitwise XOR and a comparison operator. The second method uses two comparison operators and a bitwise XOR operation is more efficient compared to a comparison operation.

We can also use following method. It doesn't use any comparison operator. The method is suggested by Hongliang and improved by gaurav.

```
 bool oppositeSigns(int x, int y)
{
    return ((x ^ y) >> 31);
}
```

The function is written only for compilers where size of an integer is 32 bit. The expression basically checks sign of (x^y) using bitwise operator '>>'. As mentioned above, the sign bit for negative numbers is always 1. The sign bit is the leftmost bit in binary representation. So we need to checks whether the 32th bit (or leftmost bit) of x^y is 1 or not. We do it by right shifting the value of x^y by 31, so that the sign bit becomes the least significant bit. If sign bit is 1, then the value of (x^y)>>31 will be 1, otherwise 0.

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/detect-if-two-integers-have-opposite-signs/

Chapter 125

Determine if a string has all Unique Characters

Determine if a string has all Unique Characters - GeeksforGeeks

Given a string, Determine if the string has all unique characters.

Examples :

```
Input : abcd10jk
Output : true

Input : hutg9mnd!nk9
Output : false
```

Approach 1 – Brute Force technique: Run 2 loops with variable i and j. Compare str[i] and str[j]. If they become equal at any point, return false.
Time Complexity: O(n^2)

C++

```
 // C++ program to illustrate string
// with unique characters using
// brute force technique
#include <bits/stdc++.h>
using namespace std;

bool uniqueCharacters(string str)
{

    // If at any time we encounter 2
```

```
    // same characters, return false
    for (int i = 0; i < str.length(); i++) {
        for (int j = i + 1; j < str.length(); j++) {
            if (str[i] == str[j]) {
                return false;
            }
        }
    }

    // If no duplicate characters encountered,
    // return true
    return true;
}

// driver code
int main()
{
    string str = "GeeksforGeeks";

    if (uniqueCharacters(str)) {
        cout << "The String " << str
             << " has all unique characters\n";
    }
    else {
        cout << "The String " << str
             << " has duplicate characters\n";
    }
    return 0;
}
// This code is contributed by Divyam Madaan
```

Java

```
 // Java program to illustrate string with
// unique characters using brute force technique
import java.util.*;

class GfG {
    boolean uniqueCharacters(String str)
    {
        // If at any time we encounter 2 same
        // characters, return false
        for (int i = 0; i < str.length(); i++)
            for (int j = i + 1; j < str.length(); j++)
                if (str.charAt(i) == str.charAt(j))
                    return false;

        // If no duplicate characters encountered,
```

```
        // return true
        return true;
    }

    public static void main(String args[])
    {
        GfG obj = new GfG();
        String input = "GeeksforGeeks";

        if (obj.uniqueCharacters(input))
            System.out.println("The String " + input + " has all unique characters");
        else
            System.out.println("The String " + input + " has duplicate characters");
    }
}
```

C#

```
 // C# program to illustrate string with
// unique characters using brute force
// technique
using System;

public class GFG {

    static bool uniqueCharacters(String str)
    {

        // If at any time we encounter 2
        // same characters, return false
        for (int i = 0; i < str.Length; i++)
            for (int j = i + 1; j < str.Length; j++)
                if (str[i] == str[j])
                    return false;

        // If no duplicate characters
        // encountered, return true
        return true;
    }

    public static void Main()
    {
        string input = "GeeksforGeeks";

        if (uniqueCharacters(input) == true)
            Console.WriteLine("The String " + input
                              + " has all unique characters");
        else
```

```
            Console.WriteLine("The String " + input
                              + " has duplicate characters");
    }
}

// This code is contributed by shiv_bhakt.
```

PHP

```
 <?php
// PHP program to illustrate string
// with unique characters using
// brute force technique

function uniqueCharacters($str)
{

    // If at any time we encounter 2
    // same characters, return false
    for($i = 0; $i < strlen($str); $i++)
    {
        for($j = $i + 1; $j < strlen($str); $j++)
        {
            if($str[$i] == $str[$j])
            {
                return false;
            }
        }
    }

    // If no duplicate characters
    // encountered, return true
    return true;
}

// Driver Code
$str = "GeeksforGeeks";

if(uniqueCharacters($str))
{
    echo "The String ", $str,
          " has all unique characters\n";
}
else
{
    echo "The String ", $str,
         " has duplicate characters\n";
}
```

```
// This code is contributed by ajit
?>
```

Output :

```
The String GeeksforGeeks has duplicate characters
```

Note: Please note that the program is case sensitive.

Approach 2 – Sorting: Using sorting based on ASCII values of characters
Time Complexity: O(n log n)

C++

```
 // C++ program to illustrate string
// with unique characters using
// brute force technique
#include <bits/stdc++.h>
using namespace std;

bool uniqueCharacters(string str)
{

    // Using sorting
    sort(str.begin(), str.end());

    for (int i = 0; i < str.length(); i++) {

        // if at any time, 2 adjacent
        // elements become equal,
        // return false
        if (str[i] == str[i + 1]) {
            return false;
        }
    }
    return true;
}

// driver code
int main()
{

    string str = "GeeksforGeeks";

    if (uniqueCharacters(str)) {
```

```
        cout << "The String " << str
             << " has all unique characters\n";
    }
    else {

        cout << "The String " << str
             << " has duplicate characters\n";
    }
    return 0;
}
// This code is contributed by Divyam Madaan
```

Java

```
 // Java program to check string with unique
// characters using sorting technique
import java.util.*;

class GfG {
    /* Convert the string to character array
       for sorting */
    boolean uniqueCharacters(String str)
    {
        char[] chArray = str.toCharArray();

        // Using sorting
        // Arrays.sort() uses binarySort in the background
        // for non-primitives which is of O(nlogn) time complexity
        Arrays.sort(chArray);

        for (int i = 0; i < chArray.length - 1; i++) {
            // if the adjacent elements are not
            // equal, move to next element
            if (chArray[i] != chArray[i + 1])
                continue;

            // if at any time, 2 adjacent elements
            // become equal, return false
            else
                return false;
        }
        return true;
    }

    // Driver code
    public static void main(String args[])
    {
        GfG obj = new GfG();
```

```
        String input = "GeeksforGeeks";

        if (obj.uniqueCharacters(input))
            System.out.println("The String " + input
              + " has all unique characters");
        else
            System.out.println("The String " + input
           + " has duplicate characters");
    }
}
```

C#

```
 // C# program to check string with unique
// characters using sorting technique
using System;

public class GFG {

    /* Convert the string to character array
    for sorting */
    static bool uniqueCharacters(String str)
    {
        char[] chArray = str.ToCharArray();

        // Using sorting
        Array.Sort(chArray);

        for (int i = 0; i < chArray.Length - 1; i++) {

            // if the adjacent elements are not
            // equal, move to next element
            if (chArray[i] != chArray[i + 1])
                continue;

            // if at any time, 2 adjacent elements
            // become equal, return false
            else
                return false;
        }

        return true;
    }

    // Driver code
    public static void Main()
    {
        string input = "GeeksforGeeks";
```

```
        if (uniqueCharacters(input) == true)
            Console.WriteLine("The String " + input
                              + " has all unique characters");
        else
            Console.WriteLine("The String " + input
                              + " has duplicate characters");
    }
}

// This code is contributed by shiv_bhakt.
```

Output:

```
The String GeeksforGeeks has duplicate characters
```

Approach 3 – Use of Extra Data Structure: This approach assumes ASCII char set(8 bits). The idea is to maintain a boolean array for the characters. The 256 indices represent 256 characters. All the array elements are initially set to false. As we iterate over the string, set true at the index equal to the int value of the character. If at any time, we encounter that the array value is already true, it means the character with that int value is repeated.

Time Complexity: O(n)

C++

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

const int MAX_CHAR = 256;

bool uniqueCharacters(string str)
{

    // If length is greater than 265,
    // some characters must have been repeated
    if (str.length() > MAX_CHAR)
        return false;

    bool chars[MAX_CHAR] = { 0 };
    for (int i = 0; i < str.length(); i++) {
        if (chars[int(str[i])] == true)
            return false;

        chars[int(str[i])] = true;
```

```
    }
    return true;
}

// driver code
int main()
{
    string str = "GeeksforGeeks";

    if (uniqueCharacters(str)) {
        cout << "The String " << str
             << " has all unique characters\n";
    }
    else {

        cout << "The String " << str
             << " has duplicate characters\n";
    }
    return 0;
}
// This code is contributed by Divyam Madaan
```

Java

```
 // Java program to illustrate String With
// Unique Characters using data structure
import java.util.*;

class GfG {
    final static MAX_CHAR = 256;

    boolean uniqueCharacters(String str)
    {
        // If length is greater than 256,
        // some characters must have been repeated
        if (str.length() > MAX_CHAR)
            return false;

        boolean[] chars = new boolean[MAX_CHAR];
        Arrays.fill(chars, false);

        for (int i = 0; i < str.length(); i++) {
            int index = (int)str.charAt(i);

            /* If the value is already true, string
               has duplicate characters, return false */
            if (chars[index] == true)
                return false;
```

```
            chars[index] = true;
        }

        /* No duplicates encountered, return true */
        return true;
    }

    // Driver code
    public static void main(String args[])
    {
        GfG obj = new GfG();
        String input = "GeeksforGeeks";

        if (obj.uniqueCharacters(input))
            System.out.println("The String " + input
                   + " has all unique characters");
        else
            System.out.println("The String " + input
                  + " has duplicate characters");
    }
}
```

Output:

```
The String  GeeksforGeeks has all unique characters
```

Approach 4 – Without Extra Data Structure: The approach is valid for strings having alphabet as a-z. This approach is little tricky. Instead of maintaining a boolean array, we maintain an integer value called checker(32 bits). As we iterate over the string, we find the int value of the character with respect to 'a' with the statement ***int bitAtIndex = str.charAt(i)-'a';***
Then the bit at that int value is set to 1 with the statement ***1 << bitAtIndex*** .
Now, if this bit is already set in the checker, the bit AND operation would make checker > 0. Return false in this case.
Else Update checker to make the bit 1 at that index with the statement ***checker = checker | (1 <<bitAtIndex);***

Time Complexity: O(n)

C++

```
 // C++ program to illustrate string
// with unique characters using
// brute force technique
#include <bits/stdc++.h>
using namespace std;
```

```
bool uniqueCharacters(string str)
{

    // Assuming string can have characters
    // a-z, this has 32 bits set to 0
    int checker = 0;

    for (int i = 0; i < str.length(); i++) {

        int bitAtIndex = str[i] - 'a';

        // if that bit is already set in
        // checker, return false
        if ((checker & (1 << bitAtIndex)) > 0) {
            return false;
        }

        // otherwise update and continue by
        // setting that bit in the checker
        checker = checker | (1 << bitAtIndex);
    }

    // no duplicates encountered, return true
    return true;
}

// driver code
int main()
{

    string str = "GeeksforGeeks";

    if (uniqueCharacters(str)) {
        cout << "The String " << str
             << " has all unique characters\n";
    }
    else {
        cout << "The String " << str
             << " has duplicate characters\n";
    }
    return 0;
}
// This code is contributed by Divyam Madaan
```

Java

```
 // Java program to illustrate String with unique
```

```
// characters without using any data structure
import java.util.*;

class GfG {
    boolean uniqueCharacters(String str)
    {
        // Assuming string can have characters a-z
        // this has 32 bits set to 0
        int checker = 0;

        for (int i = 0; i < str.length(); i++) {
            int bitAtIndex = str.charAt(i) - 'a';

            // if that bit is already set in checker,
            // return false
            if ((checker & (1 << bitAtIndex)) > 0)
                return false;

            // otherwise update and continue by
            // setting that bit in the checker
            checker = checker | (1 << bitAtIndex);
        }

        // no duplicates encountered, return true
        return true;
    }

    // Driver Code
    public static void main(String args[])
    {
        GfG obj = new GfG();
        String input = "GeekforGeeks";

        if (obj.uniqueCharacters(input))
            System.out.println("The String " + input
                               + " has all unique characters");
        else
            System.out.println("The String " + input
                               + " has duplicate characters");
    }
}
```

PHP

```
 <?php
// PHP program to illustrate
// string with unique characters
// using brute force technique
```

```
function uniqueCharacters($str)
{

    // Assuming string can have
    // characters a-z, this has
    // 32 bits set to 0
    $checker = 0;

    for ($i = 0; $i < strlen($str); $i++)
    {
        $bitAtIndex = $str[$i] - 'a';

        // if that bit is already set
        // in checker, return false
        if (($checker &
            (1 << $bitAtIndex)) > 0)
        {
            return false;
        }

    // otherwise update and continue by
    // setting that bit in the checker
    $checker = $checker |
               (1 << $bitAtIndex);
    }

    // no duplicates encountered,
    // return true
    return true;
}

// Driver Code
$str = "GeeksforGeeks";

if(uniqueCharacters($str))
{
    echo "The String ", $str,
         " has all unique characters\n";
}
else
{
    echo "The String ", $str,
         " has duplicate characters\n";
}

// This code is contributed by ajit
?>
```

Output :

```
The String GeekforGeeks has duplicate characters
```

Exercise : Above program is case sensitive, you can try making same program which is case insensitive i.e Geeks and GEeks both give similar output.

Reference:
Cracking the Coding Interview by Gayle

Improved By : shiv_bhakt, jit_t, V1

Source

https://www.geeksforgeeks.org/determine-string-unique-characters/

Chapter 126

Disjoint Set Union on trees | Set 1

Disjoint Set Union on trees | Set 1 - GeeksforGeeks

Given a tree and weights of nodes. Weights are non-negative integers. Task is to find maximum size of a subtree of a given tree such that all nodes are even in weights.

Prerequisite : Disjoint Set Union

Examples :

```
Input : Number of nodes = 7
        Weights of nodes = 1 2 6 4 2 0 3
        Edges = (1, 2), (1, 3), (2, 4),
                (2, 5), (4, 6), (6, 7)
Output : Maximum size of the subtree
with even weighted nodes = 4
Explanation :
Subtree of nodes {2, 4, 5, 6} gives the maximum size.

Input : Number of nodes = 6
        Weights of nodes = 2 4 0 2 2 6
        Edges = (1, 2), (2, 3), (3, 4),
                (4, 5), (1, 6)
Output : Maximum size of the subtree
with even weighted nodes = 6
Explanation :
The given tree gives the maximum size.
```

Approach : We can find solution by simply running DFS on tree. DFS solution gives us answer in ***O(n)***. But, how can we use DSU for this problem? We first iterate through all

edges. If both nodes are even in weights, we make union of them. Set of nodes with maximum size is the answer. If we use union-find with path compression then time complexity is ***O(n)***.

Below is the implementation of above approach :

```
 // CPP code to find maximum subtree such
// that all nodes are even in weight
#include<bits/stdc++.h>

using namespace std;

#define N 100010

// Structure for Edge
struct Edge
{
    int u, v;
};

/*
    'id': stores parent of a node.
    'sz': stores size of a DSU tree.
*/
int id[N], sz[N];

// Function to assign root
int Root(int idx)
{
    int i = idx;

    while(i != id[i])
        id[i] = id[id[i]], i = id[i];

    return i;
}

// Function to find Union
void Union(int a, int b)
{
    int i = Root(a), j = Root(b);

    if (i != j)
    {
        if(sz[i] >= sz[j])
        {
            id[j] = i, sz[i] += sz[j];
            sz[j] = 0;
        }
        else
```

```
        {
            id[i] = j, sz[j] += sz[i];
            sz[i] = 0;
        }
    }
}

// Utility function for Union
void UnionUtil(struct Edge e[], int W[], int q)
{

    for(int i = 0; i < q; i++)
    {
        // Edge between 'u' and 'v'
        int u, v;
        u = e[i].u, v = e[i].v;

        // 0-indexed nodes
        u--, v--;

        // If weights of both 'u' and 'v'
        // are even then we make union of them.
        if(W[u] % 2 == 0 && W[v] % 2 == 0)
                    Union(u,v);
    }
}

// Function to find maximum
// size of DSU tree
int findMax(int n, int W[])
{
    int maxi = 0;
    for(int i = 1; i <= n; i++)
        if(W[i] % 2 == 0)
            maxi = max(maxi, sz[i]);

    return maxi;
}

// Driver code
int main()
{
    /*
        Nodes are 0-indexed in this code
        So we have to make necessary changes
        while taking inputs
    */
```

```
    // Weights of nodes
    int W[] = {1, 2, 6, 4, 2, 0, 3};

    // Number of nodes in a tree
    int n = sizeof(W) / sizeof(W[0]);

    // Initializing every node as
    // a tree with single node.
    for(int i = 0; i < n; i++)
            id[i] = i, sz[i] = 1;

    Edge e[] = {{1, 2}, {1, 3}, {2, 4},
                {2, 5}, {4, 6}, {6, 7}};

    int q = sizeof(e) / sizeof(e[0]);

    UnionUtil(e, W, q);

    // Find maximum size of DSU tree.
    int maxi = findMax(n, W);

    printf("Maximum size of the subtree with ");
    printf("even weighted nodes = %d\n", maxi);

    return 0;
}
```

Output:

```
Maximum size of the subtree with even weighted nodes = 4
```

Source

https://www.geeksforgeeks.org/disjoint-set-union-trees-set-1/

Chapter 127

Disjoint Set Union on trees | Set 2

Disjoint Set Union on trees | Set 2 - GeeksforGeeks

Given a tree, and the cost of a subtree is defined as |S|*AND(S) where |S| is the size of the subtree and AND(S) is bitwise AND of all indices of nodes from the subtree, task is to find maximum cost of possible subtree.

Prerequisite : Disjoint Set Union

Examples:

```
Input : Number of nodes = 4
        Edges = (1, 2), (3, 4), (1, 3)
Output : Maximum cost = 4
Explanation :
Subtree with singe node {4} gives the maximum cost.

Input : Number of nodes = 6
        Edges = (1, 2), (2, 3), (3, 4), (3, 5), (5, 6)
Output : Maximum cost = 8
Explanation :
Subtree with nodes {5, 6} gives the maximum cost.
```

Approach : The strategy is to fix the AND, and find the maximum size of a subtree such that AND of all indices equals to the given AND. Suppose we fix AND as 'A'. In binary representation of A, if the ith bit is '1', then all indices(nodes) of the required subtree should have '1' in ith position in binary representation. If ith bit is '0' then indices either have '0' or '1' in ith position. That means all elements of subtree are super masks of A. All super masks of A can be generated in O(2^k) time where 'k' is the number of bits which are '0' in A.

Now, the maximum size of subtree with a given AND 'A' can be found using DSU on the tree. Let, 'u' be the super mask of A and 'p[u]' be the parent of u. If p[u] is also a super mask of A, then, we have to update the DSU by merging the components of u and p[u]. Simultaneously, we also have to keep track of the maximum size of subtree. DSU helps us to do it. It will be more clear if we look at following code.

```
 // CPP code to find maximum possible cost
#include <bits/stdc++.h>
using namespace std;

#define N 100010

// Edge structure
struct Edge {
    int u, v;
};

/* v : Adjacency list representation of Graph
    p : stores parents of nodes */
vector<int> v[N];
int p[N];

// Weighted union-find with path compression
struct wunionfind {
    int id[N], sz[N];
    void initial(int n)
    {
        for (int i = 1; i <= n; i++)
            id[i] = i, sz[i] = 1;
    }

    int Root(int idx)
    {
        int i = idx;
        while (i != id[i])
            id[i] = id[id[i]], i = id[i];

        return i;
    }

    void Union(int a, int b)
    {
        int i = Root(a), j = Root(b);
        if (i != j) {
            if (sz[i] >= sz[j]) {
                id[j] = i, sz[i] += sz[j];
                sz[j] = 0;
            }
```

```
            else {
                id[i] = j, sz[j] += sz[i];
                sz[i] = 0;
            }
        }
    }
};

wunionfind W;

// DFS is called to generate parent of
// a node from adjacency list representation
void dfs(int u, int parent)
{
    for (int i = 0; i < v[u].size(); i++) {
        int j = v[u][i];
        if (j != parent) {
            p[j] = u;
            dfs(j, u);
        }
    }
}

// Utility function for Union
int UnionUtil(int n)
{
    int ans = 0;

    // Fixed 'i' as AND
    for (int i = 1; i <= n; i++) {
        int maxi = 1;

        // Generating supermasks of 'i'
        for (int x = i; x <= n; x = (i | (x + 1))) {
            int y = p[x];

            // Checking whether p[x] is
            // also a supermask of i.
            if ((y & i) == i) {
                W.Union(x, y);

                // Keep track of maximum
                // size of subtree
                maxi = max(maxi, W.sz[W.Root(x)]);
            }
        }

        // Storing maximum cost of
```

```
        // subtree with a given AND
        ans = max(ans, maxi * i);

        // Separating components which are merged
        // during Union operation for next AND value.
        for (int x = i; x <= n; x = (i | (x + 1))) {
            W.sz[x] = 1;
            W.id[x] = x;
        }
    }

    return ans;
}

// Driver code
int main()
{
    int n, i;

    // Number of nodes
    n = 6;

    W.initial(n);

    Edge e[] = { { 1, 2 }, { 2, 3 }, { 3, 4 },
                 { 3, 5 }, { 5, 6 } };

    int q = sizeof(e) / sizeof(e[0]);

    // Taking edges as input and put
    // them in adjacency list representation
    for (i = 0; i < q; i++) {
        int x, y;
        x = e[i].u, y = e[i].v;
        v[x].push_back(y);
        v[y].push_back(x);
    }

    // Initializing parent vertex of '1' as '1'
    p[1] = 1;

    // Call DFS to generate 'p' array
    dfs(1, -1);

    int ans = UnionUtil(n);

        printf("Maximum Cost = %d\n", ans);
```

```
    return 0;
}
```

Output:

```
Maximum Cost = 8
```

Time Complexity : Union in DSU takes O(1) time. Generating all supermasks takes O(3^k) time where k is the maximum number of bits which are '0'. DFS takes O(n). Overall time complexity is O(3^k+n).

Source

https://www.geeksforgeeks.org/disjoint-set-union-trees-set-2/

Chapter 128

Divide two integers without using multiplication, division and mod operator

Divide two integers without using multiplication, division and mod operator - GeeksforGeeks

Given a two integers say a and b. Find the quotient after dividing a by b without using multiplication, division and mod operator.

Example:

```
Input : a = 10, b = 3
Output : 3

Input : a = 43, b = -8
Output :  -5
```

Approach : Keep subtracting the divisor from dividend until dividend becomes less than divisor. The dividend becomes the remainder, and the number of times subtraction is done becomes the quotient. Below is the implementation of above approach :

C++

```
 // C++ implementation to Divide two
// integers without using multiplication,
// division and mod operator
#include <bits/stdc++.h>
using namespace std;

// Function to divide a by b and
// return floor value it
```

```
int divide(int dividend, int divisor) {

  // Calculate sign of divisor i.e.,
  // sign will be negative only iff
  // either one of them is negative
  // otherwise it will be positive
  int sign = ((dividend < 0) ^ (divisor < 0)) ? -1 : 1;

  // Update both divisor and
  // dividend positive
  dividend = abs(dividend);
  divisor = abs(divisor);

  // Initialize the quotient
  int quotient = 0;
  while (dividend >= divisor) {
    dividend -= divisor;
    ++quotient;
  }

  return sign * quotient;
}

// Driver code
int main() {
  int a = 10, b = 3;
  cout << divide(a, b) << "\n";

  a = 43, b = -8;
  cout << divide(a, b);

  return 0;
}
```

Java

```
 /*Java implementation to Divide two
integers without using multiplication,
division and mod operator*/

import java.io.*;

class GFG
{

    // Function to divide a by b and
    // return floor value it
    static int divide(int dividend, int divisor)
```

```
    {

        // Calculate sign of divisor i.e.,
        // sign will be negative only iff
        // either one of them is negative
        // otherwise it will be positive
        int sign = ((dividend < 0) ^
                    (divisor < 0)) ? -1 : 1;

        // Update both divisor and
        // dividend positive
        dividend = Math.abs(dividend);
        divisor = Math.abs(divisor);

        // Initialize the quotient
        int quotient = 0;

        while (dividend >= divisor)
        {
            dividend -= divisor;
            ++quotient;
        }

        return sign * quotient;
    }

    public static void main (String[] args)
    {
        int a = 10;
        int b = 3;

        System.out.println(divide(a, b));

        a = 43;
        b = -8;

        System.out.println(divide(a, b));
    }
}

// This code is contributed by upendra singh bartwal.
```

Python3

```
 # Python 3 implementation to Divide two
# integers without using multiplication,
# division and mod operator
```

```
# Function to divide a by b and
# return floor value it
def divide(dividend, divisor):

    # Calculate sign of divisor i.e.,
    # sign will be negative only iff
    # either one of them is negative
    # otherwise it will be positive
    sign = -1 if ((dividend < 0) ^  (divisor < 0)) else 1

    # Update both divisor and
    # dividend positive
    dividend = abs(dividend)
    divisor = abs(divisor)

    # Initialize the quotient
    quotient = 0
    while (dividend >= divisor):
        dividend -= divisor
        quotient += 1


    return sign * quotient


# Driver code
a = 10
b = 3
print(divide(a, b))
a = 43
b = -8
print(divide(a, b))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# implementation to Divide two without
// using multiplication, division and mod
// operator
using System;

class GFG {

    // Function to divide a by b and
    // return floor value it
    static int divide(int dividend, int divisor)
```

```
    {

        // Calculate sign of divisor i.e.,
        // sign will be negative only iff
        // either one of them is negative
        // otherwise it will be positive
        int sign = ((dividend < 0) ^
                (divisor < 0)) ? -1 : 1;

        // Update both divisor and
        // dividend positive
        dividend = Math.Abs(dividend);
        divisor = Math.Abs(divisor);

        // Initialize the quotient
        int quotient = 0;

        while (dividend >= divisor)
        {
            dividend -= divisor;
            ++quotient;
        }

        return sign * quotient;
    }

    public static void Main ()
    {

        int a = 10;
        int b = 3;
        Console.WriteLine(divide(a, b));

        a = 43;
        b = -8;
        Console.WriteLine(divide(a, b));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP implementation to Divide two
// integers without using multiplication,
// division and mod operator
```

```
// Function to divide a by b and
// return floor value it
function divide($dividend, $divisor) {

    // Calculate sign of divisor i.e.,
    // sign will be negative only iff
    // either one of them is negative
    // otherwise it will be positive
    $sign = (($dividend < 0) ^
             ($divisor < 0)) ? -1 : 1;

    // Update both divisor and
    // dividend positive
    $dividend = abs($dividend);
    $divisor = abs($divisor);

    // Initialize the quotient
    $quotient = 0;
    while ($dividend >= $divisor)
    {
        $dividend -= $divisor;
        ++$quotient;
    }

    return $sign * $quotient;
}

// Driver code
$a = 10;
$b = 3;
echo divide($a, $b)."\n";

$a = 43;
$b = -8;
echo divide($a, $b);

// This code is contributed by Sam007
?>
```

Output :

```
3
-5
```

Time complexity : O(a)
Auxiliary space : O(1)

Efficient Approach : Use bit manipulation in order to find the quotient. The divisor and dividend can be written as

dividend = quotient * divisor + remainder

As every number can be represented in base 2(0 or 1), represent the quotient in binary form by using shift operator as given below :

1. Determine the most significant bit in the quotient. This can easily be calculated by iterating on the bit position i from 31 to 1.
2. Find the first bit for which [illegible] is less than dividend and keep updating the i^{th} bit position for which it is true.
3. Add the result in *temp* variable for checking the next position such that **(temp + (divisor << i))** is less than **dividend**.
4. Return the final answer of quotient after updating with corresponding sign.

Below is the implementation of above approach :

C++

```
 // C++ implementation to Divide two
// integers without using multiplication,
// division and mod operator
#include <bits/stdc++.h>
using namespace std;

// Function to divide a by b and
// return floor value it
int divide(long long dividend, long long divisor) {

  // Calculate sign of divisor i.e.,
  // sign will be negative only iff
  // either one of them is negative
  // otherwise it will be positive
  int sign = ((dividend < 0) ^ (divisor < 0)) ? -1 : 1;

  // remove sign of operands
  dividend = abs(dividend);
  divisor = abs(divisor);

  // Initialize the quotient
  long long quotient = 0, temp = 0;

  // test down from the highest bit and
  // accumulate the tentative value for
  // valid bit
```

```
  for (int i = 31; i >= 0; --i) {

    if (temp + (divisor << i) <= dividend) {
      temp += divisor << i;
      quotient |= 1LL << i;
    }
  }

  return sign * quotient;
}

// Driver code
int main() {
  int a = 10, b = 3;
  cout << divide(a, b) << "\n";

  a = 43, b = -8;
  cout << divide(a, b);

  return 0;
}
```

Java

```
 // Java implementation to Divide
// two integers without using
// multiplication, division
// and mod operator
import java.io.*;
import java.util.*;

// Function to divide a by b
// and return floor value it
class GFG
{
public static long divide(long dividend,
                          long divisor)
{

// Calculate sign of divisor
// i.e., sign will be negative
// only iff either one of them
// is negative otherwise it
// will be positive
long sign = ((dividend < 0) ^
             (divisor < 0)) ? -1 : 1;

// remove sign of operands
```

```
dividend = Math.abs(dividend);
divisor = Math.abs(divisor);

// Initialize the quotient
long quotient = 0, temp = 0;

// test down from the highest
// bit and accumulate the
// tentative value for
// valid bit
for (int i = 31; i >= 0; --i)
{

    if (temp + (divisor << i) <= dividend)
    {
        temp += divisor << i;
        quotient |= 1 << i;
    }
}

return (sign * quotient);
}

// Driver code
public static void main(String args[])
{
int a = 10, b = 3;
System.out.println(divide(a, b));

int a1 = 43, b1 = -8;
System.out.println(divide(a1, b1));

}
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

Python3

```
# Python3 implementation to
# Divide two integers
# without using multiplication,
# division and mod operator

# Function to divide a by
# b and return floor value it
def divide(dividend, divisor):

# Calculate sign of divisor
```

i.e., sign will be negative
either one of them is negative
only iff otherwise it will be
positive

sign = (-1 if((dividend < 0) ^ (divisor < 0)) else 1); # remove sign of operands dividend = abs(dividend); divisor = abs(divisor); # Initialize # the quotient quotient = 0; temp = 0; # test down from the highest # bit and accumulate the # tentative value for valid bit for i in range(31, -1, -1): if (temp + (divisor << i) <= dividend): temp += divisor << i; quotient |= 1 << i; return sign * quotient; # Driver code a = 10; b = 3; print(divide(a, b)); a = 43; b = -8; print(divide(a, b)); # This code is contributed by mits [tabby title="C#"]

```
 // C# implementation to Divide
// two integers without using
// multiplication, division
// and mod operator
using System;

// Function to divide a by b
// and return floor value it
class GFG
{
public static long divide(long dividend,
                          long divisor)
{

// Calculate sign of divisor
// i.e., sign will be negative
// only iff either one of them
// is negative otherwise it
// will be positive
long sign = ((dividend < 0) ^
             (divisor < 0)) ? -1 : 1;

// remove sign of operands
dividend = Math.Abs(dividend);
divisor = Math.Abs(divisor);

// Initialize the quotient
long quotient = 0, temp = 0;

// test down from the highest
// bit and accumulate the
// tentative value for
// valid bit
for (int i = 31; i >= 0; --i)
{

    if (temp + (divisor << i) <= dividend)
```

```
    {
        temp += divisor << i;
        quotient |= 1LL << i;
    }
}

return (sign * quotient);
}

// Driver code
public static void Main()
{
int a = 10, b = 3;
Console.WriteLine(divide(a, b));

int a1 = 43, b1 = -8;
Console.WriteLine(divide(a1, b1));

}
}

// This code is contributed by mits
```

PHP

```
 <?php
// PHP implementation to
// Divide two integers
// without using multiplication,
// division and mod operator

// Function to divide a by
// b and return floor value it
function divide($dividend,
                $divisor)
{

// Calculate sign of divisor
// i.e., sign will be negative
// either one of them is negative
// only iff otherwise it will be
// positive
$sign = (($dividend < 0) ^
         ($divisor < 0)) ? -1 : 1;

// remove sign of operands
$dividend = abs($dividend);
$divisor = abs($divisor);
```

```
// Initialize
// the quotient
$quotient = 0;
$temp = 0;

// test down from the highest
// bit and accumulate the
// tentative value for valid bit
for ($i = 31; $i >= 0; --$i)
{

    if ($temp + ($divisor << $i) <= $dividend)
    {
        $temp += $divisor << $i;
        $quotient |= (double)(1) << $i;
    }
}

return $sign * $quotient;
}

// Driver code
$a = 10;
$b = 3;
echo divide($a, $b). "\n";

$a = 43;
$b = -8;
echo divide($a, $b);

// This code is contributed by mits
?>
```

Output :

```
3
-5
```

Time complexity : O(log(a))
Auxiliary space : O(1)

Improved By : Sam007, Mithun Kumar, Abby_akku

Source

https://www.geeksforgeeks.org/divide-two-integers-without-using-multiplication-division-mod-operator/

Chapter 129

Divisibility by 64 with removal of bits allowed

Divisibility by 64 with removal of bits allowed - GeeksforGeeks

Given a binary string, we need to check whether that number is divisible by 64 or not after removing of some bits. If yes then print “possible” else “not possible”. We cannot make number 0 to make it divisible.

Example :

```
Input: 100010001
Output: Possible
Explanation: We can get string 1 000 000
after removing two ones which is a
representation of number 64 in the binary
numerical system.

Input: 100
Output: Not possible
Explanation : The number is 4 which is not
divisible by 64 or cannot be made possible
my removing some digits.
```

If we have 6 zeros after any one, then we can remove other bits represent it as a multiple of 64. So we just need to check if there is a 1 before six zeros.

C++

```
 // CPP program to find if given binary string can
// become divisible by 64 after removing some bits.
```

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

// function to check if it is possible
// to make it a multiple of 64.
bool checking(string s)
{
    int c = 0; // counter to count 0's
    int n = s.length(); // length of the string

    // loop which traverses right to left and
    // calculates the number of zeros before 1.
    for (int i = n - 1; i >= 0; i--) {
        if (s[i] == '0')
            c++;

        if (c >= 6 and s[i] == '1')
            return true;
    }

    return false;
}

// driver code
int main()
{
    string s = "100010001";
    if (checking(s))
        cout << "Possible";
    else
        cout << "Not possible";
    return 0;
}
```

Java

```
 // Java program to find if
// given binary string can
// become divisible by
// 64 after removing some bits
import java.io.*;

class GFG
{
    // function to check if it is possible
    // to make it a multiple of 64.
    static boolean checking(String s)
    {
```

```
        // counter to count 0's
        int c = 0;

        // length of the string
        int n = s.length();

        // loop which traverses right to left and
        // calculates the number of zeros before 1.
        for (int i = n - 1; i >= 0; i--)
        {
            if (s.charAt(i) == '0')
                c++;

            if (c >= 6 && s.charAt(i) == '1')
                return true;
        }

        return false;
    }

    // Driver code
    public static void main (String[] args)
    {
        String s = "100010001";
        if (checking(s))
            System.out.println ( "Possible");
        else
            System.out.println ( "Not possible");

    }
}

// This code  is contributed by vt_m
```

C#

```
 // C# program to find if given binary
// string can become divisible by 64
// after removing some bits
using System;

class GFG {

    // function to check if it is possible
    // to make it a multiple of 64.
    static bool checking(string s)
    {
```

```
        // counter to count 0's
        int c = 0;

        // length of the string
        int n = s.Length;

        // loop which traverses right to
        // left and calculates the number
        // of zeros before 1.
        for (int i = n - 1; i >= 0; i--)
        {
            if (s[i] == '0')
                c++;

            if (c >= 6 && s[i] == '1')
                return true;
        }

        return false;
    }

    // Driver code
    public static void Main ()
    {
        String s = "100010001";

        if (checking(s))
            Console.WriteLine (
                          "Possible");
        else
            Console.WriteLine(
                     "Not possible");
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find if
// given binary string can
// become divisible by 64
// after removing some bits.

// function to check if
// it is possible
// to make it a multiple of 64.
```

```
function checking($s)
{
    // counter to count 0's
    $c = 0;

    // length of the string
    $n = strlen($s);

    // loop which traverses right
    // to left and calculates the
    // number of zeros before 1.
    for ($i = $n - 1; $i >= 0; $i--)
    {
        if ($s[$i] == '0')
            $c++;

        if ($c >= 6 and $s[$i] == '1')
            return true;
    }

    return false;
}

// Driver Code
$s = "100010001";
if (checking($s))
    echo "Possible";
else
    echo "Not possible";

// This code is contributed by ajit
?>
```

Output :

```
Possible
```

Time Complexity: O(length of string)

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/divisibility-64-removal-bits-allowed/

Chapter 130

Efficient method for 2's complement of a binary string

Efficient method for 2's complement of a binary string - GeeksforGeeks

Given a Binary Number as string, print its 2's complements.

2's complement of a binary number is 1 added to the 1's complement of the binary number. Note that 1's complement is simply flip of given binary number.
Examples:

```
2's complement of "0111" is  "1001"
2's complement of "1100" is  "0100"
```

We have discussed 1's and 2's complements in below post.

1's and 2's complement of a Binary Number.

The method discussed in above post traverses binary string twice to find 2's complement, first finds 1's complement, then finds 2's complement using 1's complement

In this post an efficient method for 2's complement is discussed that **traverses string only once**. We traverse the string from last till the single 1 is not traversed and after that flip all values of string i.e. 0 to 1 and 1 to 0.

Note: Here to handle the corner case i.e. if 1 doesn't exist in the string then just append 1 in the starting of string.

Illustration :

```
Input:  str = "1000100"
Output:       0111100
Explanation: Starts traversing the string from last,
```

```
we got first '1' at index 4 then just flip the bits
of 0 to 3 indexes to make the 2's complement.

Input:  str =  "0000"
Output:        10000
Explanation: As there is no 1 in the string so just
append '1' at starting.
```

C++

```
 // An efficient C++ program to find 2's complement
#include<bits/stdc++.h>
using namespace std;

// Function to find two's complement
string findTwoscomplement(string str)
{
    int n = str.length();

    // Traverse the string to get first '1' from
    // the last of string
    int i;
    for (i = n-1 ; i >= 0 ; i--)
        if (str[i] == '1')
            break;

    // If there exists no '1' concatenate 1 at the
    // starting of string
    if (i == -1)
        return '1' + str;

    // Continue traversal after the position of
    // first '1'
    for (int k = i-1 ; k >= 0; k--)
    {
        //Just flip the values
        if (str[k] == '1')
            str[k] = '0';
        else
            str[k] = '1';
    }

    // return the modified string
    return str;;
}

// Driver code
int main()
```

```
{
    string str = "00000101";
    cout << findTwoscomplement(str);
    return 0;
}
```

Java

```
 // An efficient Java program to find 2's complement

class Test
{
    // Method to find two's complement
    static String findTwoscomplement(StringBuffer str)
    {
        int n = str.length();

        // Traverse the string to get first '1' from
        // the last of string
        int i;
        for (i = n-1 ; i >= 0 ; i--)
            if (str.charAt(i) == '1')
                break;

        // If there exists no '1' concat 1 at the
        // starting of string
        if (i == -1)
            return "1" + str;

        // Continue traversal after the position of
        // first '1'
        for (int k = i-1 ; k >= 0; k--)
        {
            //Just flip the values
            if (str.charAt(k) == '1')
                str.replace(k, k+1, "0");
            else
                str.replace(k, k+1, "1");
        }

        // return the modified string
        return str.toString();
    }

    // Driver method
    public static void main(String[] args)
    {
        StringBuffer str = new StringBuffer("00000101");
```

```
        System.out.println(findTwoscomplement(str));
    }
}
```

PHP

```
 <?php
// An efficient PHP program to find 2's
// complement

// Function to find two's complement
function findTwoscomplement($str)
{
    $n = strlen($str);

    // Traverse the string to get first
    // '1' from the last of string
    $i;
    for ($i = $n-1 ; $i >= 0 ; $i--)
        if ($str[$i] == '1')
            break;

    // If there exists no '1' concatenate
    // 1 at the starting of string
    if ($i == -1)
        return '1' + $str;

    // Continue traversal after the
    // position of first '1'
    for ($k = $i-1 ; $k >= 0; $k--)
    {
        // Just flip the values
        if ($str[$k] == '1')
            $str[$k] = '0';
        else
            $str[$k] = '1';
    }

    // return the modified string
    return $str;;
}

// Driver code
$str = "00000101";
echo findTwoscomplement($str);

// This code is contributed by jit.
?>
```

Output:

```
11111011
```

Improved By : bois, jit_t

Source

https://www.geeksforgeeks.org/efficient-method-2s-complement-binary-string/

Chapter 131

Efficient way to multiply with 7

Efficient way to multiply with 7 - GeeksforGeeks

We can multiply a number by 7 using bitwise operator. First left shift the number by 3 bits (you will get 8n) then subtract the original numberfrom the shifted number and return the difference (8n – n).

Program:

C

```
 # include<stdio.h>

int multiplyBySeven(unsigned int n)
{
    /* Note the inner bracket here. This is needed
       because precedence of '-' operator is higher
       than '<<' */
    return ((n<<3) - n);
}

/* Driver program to test above function */
int main()
{
    unsigned int n = 4;
    printf("%u", multiplyBySeven(n));

    getchar();
    return 0;
}
```

Java

```
 // Java program to multiply any
// positive number to 7

class GFG {

    static int multiplyBySeven(int n)
    {
        /* Note the inner bracket here.
        This is needed because precedence
        of '-' operator is higher
        than '<<' */
        return ((n << 3) - n);
    }

    // Driver code
    public static void main (String arg[])
    {
        int n = 4;

        System.out.println(multiplyBySeven(n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python

```
 # Python program to multiply any
# positive number to 7

# Function to mutiply any number with 7
def multiplyBySeven(n):

    # Note the inner bracket here.
    # This is needed because
    # precedence of '-' operator is
    # higher than '<<'
    return ((n << 3) - n)

# Driver code
n = 4
print(multiplyBySeven(n))

# This code is contributed by Danish Raza
```

C#

```
 // C# program to multiply any
```

```
// positive number to 7
using System;

class GFG
{
    static int multiplyBySeven(int n)
    {
        /* Note the inner bracket here. This is needed
        because precedence of '-' operator is higher
        than '<<' */
        return ((n << 3) - n);
    }

    // Driver code
    public static void Main ()
    {
        int n = 4;
        Console.Write(multiplyBySeven(n));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php


function multiplyBySeven($n)
{

    // Note the inner bracket here.
    // This is needed because
    // precedence of '-' operator
    // is higherthan '<<'
    return (($n<<3) - $n);
}

    // Driver Code
    $n = 4;
    echo multiplyBySeven($n);

// This code is contributed by vt_m.
?>
```

Output:

```
28
```

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

Note: Works only for positive integers.
Same concept can be used for fast multiplication by 9 or other numbers.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/efficient-way-to-multiply-with-7/

Chapter 132

Efficiently check if a string has duplicates without using any additional data structure

Efficiently check if a string has duplicates without using any additional data structure - GeeksforGeeks

Implement an space efficient algorithm to determine if a string (of characters from 'a' to 'z') has all unique characters or not. Use additional data structures like count array, hash, etc is not allowed.

Expected Time Complexity : O(n)
Examples :

```
Input  : str = "aaabbccdaa"
Output : No

Input  : str = "abcd"
Output : Yes
```

The idea is to use an integer variable and use bits in its binary representation to store whether a character is present or not. Typically an integer has at-least 32 bits and we need to store presence/absence of only 26 characters.

Below is the implementation of the idea.

C++

```
 // A space efficient C++ program to check if
// all characters of string are unique.
#include<bits/stdc++.h>
```

```
using namespace std;

// Returns true if all characters of str are
// unique.
// Assumptions : (1) str contains only characters
//                   from 'a' to 'z'
//               (2) integers are stored using 32
//                   bits
bool areChractersUnique(string str)
{
    // An integer to store presence/absence
    // of 26 characters using its 32 bits.
    int checker = 0;

    for (int i = 0; i < str.length(); ++i)
    {
        int val = (str[i]-'a');

        // If bit corresponding to current
        // character is already set
        if ((checker & (1 << val)) > 0)
            return false;

        // set bit in checker
        checker |=  (1 << val);
    }

    return true;
}

// Driver code
int main()
{
    string s = "aaabbccdaa";
    if (areChractersUnique(s))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // A space efficient Java program to check if
// all characters of string are unique.
class GFG {

    // Returns true if all characters of str are
```

```
    // unique.
    // Assumptions : (1) str contains only characters
    //                   from 'a' to 'z'
    //               (2) integers are stored using 32
    //                   bits
    static boolean areChractersUnique(String str)
    {

        // An integer to store presence/absence
        // of 26 characters using its 32 bits.
        int checker = 0;

        for (int i = 0; i < str.length(); ++i)
        {
            int val = (str.charAt(i)-'a');

            // If bit corresponding to current
            // character is already set
            if ((checker & (1 << val)) > 0)
                return false;

            // set bit in checker
            checker |= (1 << val);
        }

        return true;
    }

    //driver code
    public static void main (String[] args)
    {
        String s = "aaabbccdaa";

        if (areChractersUnique(s))
            System.out.print("Yes");
        else
            System.out.print("No");
    }
}

// This code is contributed by Anant Agarwal.
```

C#

```
 // A space efficient program
// to check if all characters
// of string are unique.
using System;
```

```
class GFG {

    // Returns true if all characters
    // of str are unique. Assumptions:
    // (1)str contains only characters
    // from 'a' to 'z'.(2)integers are
    // stored using 32 bits
    static bool areChractersUnique(string str)
    {
        // An integer to store presence
        // or absence of 26 characters
        // using its 32 bits.
        int checker = 0;

        for (int i = 0; i < str.Length; ++i) {
            int val = (str[i] - 'a');

            // If bit corresponding to current
            // character is already set
            if ((checker & (1 << val)) > 0)
                return false;

            // set bit in checker
            checker |= (1 << val);
        }

        return true;
    }

    // Driver code
    public static void Main()
    {
        string s = "aaabbccdaa";

        if (areChractersUnique(s))
            Console.Write("Yes");
        else
            Console.Write("No");
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// A space efficient PHP program
```

```
// to check if all characters of
// string are unique.

// Returns true if all characters
// of str are unique.
// Assumptions : (1) str contains
//                     only characters
//                   from 'a' to 'z'
//         (2) integers are stored
//             using 32 bits
function areChractersUnique($str)
{
    // An integer to store presence/absence
    // of 26 characters using its 32 bits.
    $checker = 0;

    for ($i = 0; $i < $len = strlen($str); ++$i)
    {
        $val = ($str[$i] - 'a');

        // If bit corresponding to current
        // character is already set
        if (($checker & (1 << $val)) > 0)
            return false;

        // set bit in checker
        $checker |= (1 << $val);
    }

    return true;
}

// Driver code
$s = "aaabbccdaa";
if (areChractersUnique($s))
    echo "Yes";
else
    echo "No";

// This code is contributed by aj_36
?>
```

Output :

```
No
```

Time Complexity : O(n)

Auxiliary Space : O(1)

Improved By : jit_t

Source

https://www.geeksforgeeks.org/efficiently-check-string-duplicates-without-using-additional-data-structure/

Chapter 133

Efficiently check whether n is a multiple of 4 or not

Efficiently check whether n is a multiple of 4 or not - GeeksforGeeks

Given a number **n**. The problem is to efficiently check whether **n** is a multiple of 4 or not without using arithmetic operators.

Examples:

```
Input : 16
Output : Yes

Input : 14
Output : No
```

Approach: A multiple of 4 always has **00** as its last two digits in its binary representation. We have to check whether the last two digits of **n** are unset or not.

How to check whether the last two bits are unset or not.
If n & 3 == 0, then the last two bits are unset, else either both or one of them are set.

C/C++

```
 // C++ implementation to efficiently check whether n
// is a multiple of 4 or not
#include <bits/stdc++.h>
using namespace std;

// function to check whether 'n' is
// a multiple of 4 or not
string isAMultipleOf4(int n)
{
```

```
    // if true, then 'n' is a multiple of 4
    if ((n & 3) == 0)
        return "Yes";

    // else 'n' is not a multiple of 4
    return "No";
}

// Driver program to test above
int main()
{
    int n = 16;
    cout << isAMultipleOf4(n);
    return 0;
}
```

Java

```
 // Java implementation to efficiently check
// whether n is a multiple of 4 or not

class GFG {
    // method to check whether 'n' is
    // a multiple of 4 or not
    static boolean isAMultipleOf4(int n)
    {
        // if true, then 'n' is a multiple of 4
        if ((n & 3) == 0)
            return true;

        // else 'n' is not a multiple of 4
        return false;
    }

    // Driver method
    public static void main(String[] args)
    {
        int n = 16;
        System.out.println(isAMultipleOf4(n) ? "Yes" : "No");
    }
}
```

C#

```
 // C# implementation to efficiently check
// whether n is a multiple of 4 or not
using System;
```

```
class GFG {

    // method to check whether 'n' is
    // a multiple of 4 or not
    static bool isAMultipleOf4(int n)
    {
        // if true, then 'n' is a multiple of 4
        if ((n & 3) == 0)
            return true;

        // else 'n' is not a multiple of 4
        return false;
    }

    // Driver method
    public static void Main()
    {
        int n = 16;
        Console.WriteLine(isAMultipleOf4(n) ? "Yes" : "No");
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP implementation to
// efficiently check whether n
// is a multiple of 4 or not

// function to check whether 'n' is
// a multiple of 4 or not
function isAMultipleOf4($n)
{

    // if true, then 'n'
    // is a multiple of 4
    if (($n & 3) == 0)
        return "Yes";

    // else 'n' is not
    // a multiple of 4
    return "No";
}

    // Driver Code
```

```
    $n = 16;
    echo isAMultipleOf4($n);

// This code is contributed by anuj_67.
?>
```

Output:

```
Yes
```

Can we generalize above solution?
Similarly we can check for other powers of 2. For example, a number n would be multiple of 8 if n & 7 is 0. In general we can say.

```
// x must be a power of 2 for below
// logic to work
if (n & (x -1) == n)
   n is a multiple of x
Else
   n is NOT a multiple of x
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/efficiently-check-whether-n-multiple-4-not/

Chapter 134

Efficiently find first repeated character in a string without using any additional data structure in one traversal

Efficiently find first repeated character in a string without using any additional data structure in one traversal - GeeksforGeeks

Implement a space efficient algorithm to check First repeated character in a string without using any additional data structure in one traversal. Use additional data structures like count array, hash, etc is not allowed.

Examples :

```
Input :  abcfdeacf
Output : char = a, index= 6
```

The idea is to use an integer variable and uses bits in its binary representation to store whether a character is present or not. Typically an integer has at-least 32 bits and we need to store presence/absence of only 26 characters.

C++

```
 // Efficiently check First repeated character
// in C++ program
#include<bits/stdc++.h>
using namespace std;

// Returns -1 if all characters of str are
```

```
// unique.
// Assumptions : (1) str contains only characters
//                   from 'a' to 'z'
//               (2) integers are stored using 32
//                   bits
int FirstRepeated(string str)
{
    // An integer to store presence/absence
    // of 26 characters using its 32 bits.
    int checker = 0;

    for (int i = 0; i < str.length(); ++i)
    {
        int val = (str[i]-'a');

        // If bit corresponding to current
        // character is already set
        if ((checker & (1 << val)) > 0)
            return i;

        // set bit in checker
        checker |= (1 << val);
    }

    return -1;
}

// Driver code
int main()
{
    string s = "abcfdeacf";
    int i=FirstRepeated(s);
    if (i!=-1)
        cout <<"Char = "<< s[i] << "   and Index = "<<i;
    else
        cout << "No repeated Char";
    return 0;
}
```

Java

```
 // Efficiently check First repeated character
// in Java program
public class First_Repeated_char {

    static int FirstRepeated(String str)
    {
        // An integer to store presence/absence
```

```
        // of 26 characters using its 32 bits.
        int checker = 0;

        for (int i = 0; i < str.length(); ++i)
        {
            int val = (str.charAt(i)-'a');

            // If bit corresponding to current
            // character is already set
            if ((checker & (1 << val)) > 0)
                return i;

            // set bit in checker
            checker |= (1 << val);
        }

        return -1;
    }

    // Driver code
    public static void main(String args[])
    {
        String s = "abcfdeacf";
        int i=FirstRepeated(s);
        if (i!=-1)
           System.out.println("Char = "+ s.charAt(i) + "   and Index = "+i);
        else
            System.out.println( "No repeated Char");
    }
}
// This code is contributed by Sumit Ghosh
```

Python

```
 # Efficiently check First repeated character
# in Python

# Returns -1 if all characters of str are
# unique.
# Assumptions : (1) str contains only characters
#                 from 'a' to 'z'
##             (2) integers are stored using 32
##                 bits
def FirstRepeated(string):

    # An integer to store presence/absence
    # of 26 characters using its 32 bits.
    checker = 0
```

```
    pos = 0
    for i in string:
        val = ord(i) - ord('a');

        # If bit corresponding to current
        # character is already set
        if ((checker & (1 << val)) > 0):
            return pos

        # set bit in checker
        checker |= (1 << val)
        pos += 1

    return -1

# Driver code
string = "abcfdeacf"
i = FirstRepeated(string)
if i != -1:
    print "Char = ", string[i], " and Index = ", i;
else:
    print "No repeated Char"

# This code is contributed by Sachin Bisht
```

C#

```
 // C# program to Efficiently
// check First repeated character
using System;

public class First_Repeated_char {

    static int FirstRepeated(string str)
    {
        // An integer to store presence/absence
        // of 26 characters using its 32 bits.
        int checker = 0;

        for (int i = 0; i < str.Length; ++i)
        {
            int val = (str[i]-'a');

            // If bit corresponding to current
            // character is already set
            if ((checker & (1 << val)) > 0)
                return i;
```

```
            // set bit in checker
            checker |= (1 << val);
        }

        return -1;
    }

    // Driver code
    public static void Main()
    {
        string s = "abcfdeacf";
        int i=FirstRepeated(s);
        if (i!=-1)
           Console.WriteLine("Char = " + s[i] +
                          " and Index = " + i);
        else
            Console.WriteLine( "No repeated Char");
    }
}

// This code is contributed by vt_m.
```

Output:

```
Char = a   and Index = 6
```

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

Improved By : vt_m

Source

https://www.geeksforgeeks.org/efficiently-find-first-repeated-character-string-without-using-additional-data-structu

Chapter 135

Equal Sum and XOR

Equal Sum and XOR - GeeksforGeeks

Given a positive integer n, find count of positive integers i such that 0 <= i <= n and n+i = n^i
Examples :

```
Input  : n = 7
Output : 1
Explanation:
7^i = 7+i holds only for only for i = 0
7+0 = 7^0 = 7
```

```
Input n = 12
Output: 4
12^i = 12+i hold only for i = 0, 1, 2, 3
for i=0, 12+0 = 12^0 = 12
for i=1, 12+1 = 12^1 = 13
for i=2, 12+2 = 12^2 = 14
for i=3, 12+3 = 12^3 = 15
```

Method 1 (Simple) :
One simple solution is to iterate over all values of i 0<= i <= n and count all satisfying values.

C++

```
 /* C++ program to print count of values such
   that n+i = n^i */
#include <iostream>
```

```
using namespace std;

// function to count number of values less than
// equal to n that satisfy the given condition
int countValues (int n)
{
    int countV = 0;

    // Traverse all numbers from 0 to n and
    // increment result only when given condition
    // is satisfied.
    for (int i=0; i<=n; i++ )
        if ((n+i) == (n^i) )
            countV++;

    return countV;
}

// Driver program
int main()
{
    int n = 12;
    cout << countValues(n);
    return 0;
}
```

Java

```
 /* Java program to print count of values
 such that n+i = n^i */
import java.util.*;

class GFG {

    // function to count number of values
    // less than equal to n that satisfy
    // the given condition
    public static int countValues (int n)
    {
        int countV = 0;

        // Traverse all numbers from 0 to n
        // and increment result only when
        // given condition is satisfied.
        for (int i = 0; i <= n; i++ )
            if ((n + i) == (n ^ i) )
                countV++;
```

```
        return countV;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int n = 12;
        System.out.println(countValues(n));

    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python3 program to print count
# of values such that n+i = n^i

# function to count number
# of values less than
# equal to n that satisfy
# the given condition
def countValues (n):
    countV = 0;

    # Traverse all numbers
    # from 0 to n and
    # increment result only
    # when given condition
    # is satisfied.
    for i in range(n + 1):
        if ((n + i) == (n ^ i)):
            countV += 1;

    return countV;

# Driver Code
n = 12;
print(countValues(n));

# This code is contributed by mits
```

C#

```
 /* C# program to print count of values
such that n+i = n^i */
```

```
using System;

class GFG {

    // function to count number of values
    // less than equal to n that satisfy
    // the given condition
    public static int countValues (int n)
    {
        int countV = 0;

        // Traverse all numbers from 0 to n
        // and increment result only when
        // given condition is satisfied.
        for (int i = 0; i <= n; i++ )
            if ((n + i) == (n ^ i) )
                countV++;

        return countV;
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int n = 12;
        Console.WriteLine(countValues(n));

    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to print count
// of values such that n+i = n^i

// function to count number
// of values less than
// equal to n that satisfy
// the given condition
function countValues ($n)
{
    $countV = 0;

    // Traverse all numbers
    // from 0 to n and
```

```
    // increment result only
    // when given condition
    // is satisfied.
    for ($i = 0; $i <= $n; $i++ )
        if (($n + $i) == ($n^$i) )
            $countV++;

    return $countV;
}

    // Driver Code
    $n = 12;
    echo countValues($n);

// This code is contributed by m_kit
?>
```

Output:

```
4
```

Method 2 (Efficient) :
An efficient solution is as follows

Since we know a + b = a ^ b + 2 * (a & b)

We can write, n + i = n ^ i + 2 * (n & i)

So n + i = n ^ i implies n & i = 0

Hence our problem reduces to finding values of i such that n & i = 0. How to find count of such pairs? We can use the count of unset-bits in the binary representation of n. For n & i to be zero, i must unset all set-bits of n. If the kth bit is set at a particular in n, kth bit in i must be 0 always, else kth bit of i can be 0 or 1

Hence, total such combinations are 2^(count of unset bits in n)

For example, consider n = 12 (Binary representation : 1 1 0 0).
All possible values of i that can unset all bits of n are 0 0 0/1 0/1 where 0/1 implies either 0 or 1. Number of such values of i are 2^2 = 4.

The following is the program following the above idea.

C++

```
 /* c++ program to print count of values such
  that n+i = n^i */
#include <bits/stdc++.h>
using namespace std;
```

```
// function to count number of values less than
// equal to n that satisfy the given condition
int countValues(int n)
{
    // unset_bits keeps track of count of un-set
    // bits in binary representation of n
    int unset_bits=0;
    while (n)
    {
        if ((n & 1) == 0)
            unset_bits++;
        n=n>>1;
    }

    // Return 2 ^ unset_bits
    return 1 << unset_bits;
}

// Driver code
int main()
{
    int n = 12;
    cout << countValues(n);
    return 0;
}
```

Java

```
 /* Java program to print count of values
  such that n+i = n^i */
import java.util.*;

class GFG {

    // function to count number of values
    // less than equal to n that satisfy
    // the given condition
    public static int countValues(int n)
    {
        // unset_bits keeps track of count
        // of un-set bits in binary
        // representation of n
        int unset_bits=0;
        while (n > 0)
        {
            if ((n & 1) == 0)
                unset_bits++;
```

```
            n=n>>1;
        }

        // Return 2 ^ unset_bits
        return 1 << unset_bits;
    }

    /* Driver program to test above
    function */
    public static void main(String[] args)
    {
        int n = 12;
        System.out.println(countValues(n));

    }
}

// This code is contributed by Arnav Kr. Mandal.
```

C#

```
 // C# program to print count of
// values such that n+i = n^i
using System;

class GFG
{

    // function to count number of
    // values less than equal to n
    // that satisfy the given condition
    static int countValues(int n)
    {
        // unset_bits keeps track of
        // count of un-set bits in
        // binary representation of n
        int unset_bits = 0;
        while (n > 0)
        {
            if ((n & 1) == 0)
                unset_bits++;
            n = n >> 1;
        }

        // Return 2 ^ unset_bits
        return 1 << unset_bits;
    }
```

```
    // Driver Code
    public static void Main()
    {
        int n = 12;
        Console.Write(countValues(n));
    }
}

// This code is contributed by Anuj_67.
```

PHP

```
 <?php
/* PHP program to print count
of values such that n+i = n^i */


// function to count number of
// values less than equal to n
// that satisfy the given
// condition
function countValues( $n)
{

    // unset_bits keeps track
    // of count of un-set bits
    // in binary representation
    // of n
    $unset_bits = 0;
    while ($n)
    {
        if (($n & 1) == 0)
            $unset_bits++;
        $n = $n >> 1;
    }

    // Return 2 ^ unset_bits
    return 1 << $unset_bits;
}

// Driver code

    $n = 12;
    echo countValues($n);

// This code is contributed
// by Anuj_67.
?>
```

Output :

```
4
```

Improved By : jit_t, vt_m, Mithun Kumar

Source

https://www.geeksforgeeks.org/equal-sum-xor/

Chapter 136

Euclid's Algorithm when % and / operations are costly

Euclid's Algorithm when % and / operations are costly - GeeksforGeeks

Euclid's algorithmis used to find GCD of two numbers.

There are mainly two versions of algorithm.
Version 1 (Using subtraction)

```
 // Recursive function to return gcd of a and b
int gcd(int a, int b)
{
    if (a == b)
       return a;

    return (a > b)? gcd(a-b, b): gcd(a, b-a);
}
```

Version 2 (Using modulo operator)

```
 // Function to return gcd of a and b
int gcd(int a, int b)
{
    if (a == 0)
       return b;

    return gcd(b%a, a);
}
```

Which of the above two is more efficient?
Version 1 can take linear time to find the GCD, consider the situation when one of the given

numbers is much bigger than the other. Version 2 is obviously more efficient as there are less recursive calls and takes logarithmic time.

Consider a situation where modulo operator is not allowed, can we optimize version 1 to work faster?

Below are some important observations. The idea is to use bitwise operators. We can find x/2 using x>>1. We can check whether x is odd or even using x&1.

gcd(a, b) = 2*gcd(a/2, b/2) if both a and b are even.
gcd(a, b) = gcd(a/2, b) if a is even and b is odd.
gcd(a, b) = gcd(a, b/2) if a is odd and b is even.

Below is C++ implementation.

```
 // Efficient C++ program when % and / are not allowed
int gcd(int a, int b)
{
    // Base cases
    if (b == 0 || a == b) return a;
    if (a == 0) return b;

    // If both a and b are even, divide both a
    // and b by 2.  And multiply the result with 2
    if ( (a & 1) == 0 && (b & 1) == 0 )
       return gcd(a>>1, b>>1) << 1;

    // If a is even and b is odd, divide a by 2
    if ( (a & 1) == 0 && (b & 1) != 0 )
       return gcd(a>>1, b);

    // If a is odd and b is even, divide b by 2
    if ( (a & 1) != 0 && (b & 1) == 0 )
       return gcd(a, b>>1);

    // If both are odd, then apply normal subtraction
    // algorithm.  Note that odd-odd case always
    // converts odd-even case after one recursion
    return (a > b)? gcd(a-b, b): gcd(a, b-a);
}
```

This article is compiled by Shivam Agrawal. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/euclids-algorithm-when-and-operations-are-costly/

Chapter 137

Extract ‘k’ bits from a given position in a number.

Extract ’k’ bits from a given position in a number. - GeeksforGeeks

How to extract ‘k’ bits from a given position ‘p’ in a number?

Examples:

```
Input : number = 171
             k = 5
             p = 2
Output : The extracted number is 21
171 is represented as 0101011 in binary,
so, you should get only 10101 i.e. 21.

Input : number = 72
            k = 5
            p = 1
Output : The extracted number is 8
72 is represented as 1001000 in binary,
so, you should get only 01000 i.e 8.
```

1) Right shift number by p-1.
2) Do bit wise AND of k set bits with the modified number. We can get k set bits by doing (1 << k) – 1.

C

```
 // C program to extract k bits from a given
// position.
```

```
#include <stdio.h>

// Function to extract k bits from p position
// and returns the extracted value as integer
int bitExtracted(int number, int k, int p)
{
    return (((1 << k) - 1) & (number >> (p - 1)));
}

// Driver code
int main()
{
    int number = 171, k = 5, p = 2;
    printf("The extracted number is %d",
               bitExtracted(number, k, p));
    return 0;
}
```

Java

```
 // Java program to extract k bits from a given
// position.

class GFG {

    // Function to extract k bits from p position
    // and returns the extracted value as integer
    static int bitExtracted(int number, int k, int p)
    {
        return (((1 << k) - 1) & (number >> (p - 1)));
    }

    // Driver code
    public static void main (String[] args) {
        int number = 171, k = 5, p = 2;
        System.out.println("The extracted number is "+
               bitExtracted(number, k, p));
    }
}
```

Python

```
 # Python program to extract k bits from a given
# position.

# Function to extract k bits from p position
# and returns the extracted value as integer
```

```
def bitExtracted(number, k, p):
    return ( ((1 << k) - 1)  &  (number >> (p-1) ) );

# number is from where 'k' bits are extracted
# from p position
number = 171
k = 5
p = 2
print "The extracted number is ", bitExtracted(number, k, p)
```

C#

```
 // C# program to extract k bits from a given
// position.
using System;

class GFG {

    // Function to extract k bits from p position
    // and returns the extracted value as integer
    static int bitExtracted(int number, int k, int p)
    {
        return (((1 << k) - 1) & (number >> (p - 1)));
    }

    // Driver code
    public static void Main()
    {
        int number = 171, k = 5, p = 2;

        Console.WriteLine("The extracted number is "
                      + bitExtracted(number, k, p));
    }
}

//This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
//PHP program to extract
// k bits from a given
// position.

// Function to extract k
// bits from p position
// and returns the extracted
```

```
// value as integer
function bitExtracted($number, $k, $p)
{
    return (((1 << $k) - 1) &
           ($number >> ($p - 1)));
}

    // Driver Code
    $number = 171; $k = 5; $p = 2;
    echo "The extracted number is ",
          bitExtracted($number, $k, $p);

// This code is contributed by Ajit
?>
```

Output :

```
The extracted number is 21
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/extract-k-bits-given-position-number/

Chapter 138

Fast average of two numbers without division

Fast average of two numbers without division - GeeksforGeeks

Given two numbers, find floor of their average without using division.

```
Input : x = 10, y = 12
Output : 11

Input : x = 10, y = 7
Output : 8
We take floor of sum.
```

The idea is to use right shift operator, instead of doing (x + y)/2, we do (x + y) >> 1

C

```
 // C program to find average without using
// division.
#include <stdio.h>

int floorAvg(int x, int y) {
     return (x + y) >> 1; }

int main() {
  int x = 10, y = 20;
  printf("\n\nAverage = %d\n\n", floorAvg(x, y));
  return 0;
}
```

Java

```
 // Java program to find average
// without using division
class GFG
{
    static int floorAvg(int x, int y) {
    return (x + y) >> 1; }

    // Driver code
    public static void main (String[] args)
    {
        int x = 10, y = 20;
        System.out.print(floorAvg(x, y));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find average
# without using division.
def floorAvg(x, y):
    return (x + y) >> 1

# Driver code
x = 10
y = 20

print("Average ", floorAvg(x, y))

# This code is contributed by sunny singh
```

C#

```
 // C# program to find average
// without using division
using System;

class GFG
{
    static int floorAvg(int x, int y)
    {
      return (x + y) >> 1;
    }
```

```
    // Driver code
    public static void Main (String[] args)
    {
        int x = 10, y = 20;
        Console.Write("Average = ");
        Console.Write(floorAvg(x, y));
    }
}

// This code is contributed by parashar...
```

PHP

```
 <?php
// PHP program to find average
// without using division.

// function returns
// the average
function floorAvg($x, $y)
{
    return ($x + $y) >> 1;

}

    // Driver Code
    $x = 10;
    $y = 20;
    echo "Average = ", floorAvg($x, $y);

// This Code is contributed by Ajit
?>
```

Output:

```
Average = 15
```

Applications :
We need floor of average of two numbers in many standard algorithms like Merge Sort,Binary Search, etc. Since we use bitwise operator instead of division, the above way of finding average is faster.

Improved By : parashar, Dildar_Sk, jit_t

Source

https://www.geeksforgeeks.org/fast-average-two-numbers-without-division/

Chapter 139

Fast inverse square root

Fast inverse square root - GeeksforGeeks

Fast inverse square root is an algorithm that estimates $\frac{1}{\sqrt{x}}$, the reciprocal (or multiplicative inverse) of the square root of a 32-bit floating-point number x in IEEE 754 floating-point format. Computing reciprocal square roots is necessary in many applications, such as vector normalization in video games and is mostly used in calculations involved in 3D programming. In 3D graphics, surface normals, 3-coordinate vectors of length 1 is used, to express lighting and reflection. There were a lot of surface normals. And calculating them involves normalizing a lot of vectors. Normalizing is often just a fancy term for division. The Pythagorean theorem computes distance between points, and dividing by distance helps normalize vectors:

$$\frac{1}{\sqrt{x^2+y^2+z^2}}$$

This algorithm is best known for its implementation in 1999 in the source code of Quake III Arena Game, a first-person shooter video game that made heavy use of 3D graphics. At that time, it was generally computationally expensive to compute the reciprocal of a floating-point number, especially on a large scale; the fast inverse square root bypassed this step.

Algorithm :
Step 1 : It reinterprets the bits of the floating-point input as an integer.

```
i  = * ( long * ) &y;
```

Step 2 : It takes the resulting value and does integer arithmetic on it which produces an approximation of the value we're looking for.

```
i  = 0x5f3759df - ( i >> 1 );
```

Step 3 : The result is not the approximation itself though, it is an integer which happens to be, if you reinterpret the bits as a floating point number, the approximation. So the code does the reverse of the conversion in step 1 to get back to floating point:

```
y  = * ( float * ) &i;
```

Step 4 : And finally it runs a single iteration of Newton's method to improve the approximation.

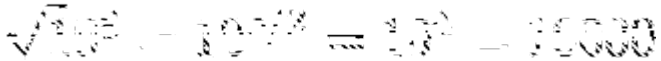
```
y  = y * ( threehalfs - ( x2 * y * y ) );         //threehalfs = 1.5F;
```

The algorithm accepts a 32-bit floating-point number as the input and stores a halved value for later use. Then, treating the bits representing the floating-point number as a 32-bit integer, a logical shift right by one bit is performed and the result subtracted from the magic number 0x5F3759DF. This is the first approximation of the inverse square root of the input. Treating the bits again as a floating-point number, it runs one iteration of Newton's approximation method, yielding a more precise approximation.

Let's say there is a number in exponent form or scientific notation:

10^8 =100 million

Now, to find the regular square root, we'd just divide the exponent by 2:



And if, want to know the inverse square root, divide the exponent by -2 to flip the sign:

So, the code converts the floating-point number into an integer. It then shifts the bits by one, which means the exponent bits are divided by 2 (when we eventually turn the bits back into a float). And lastly, to negate the exponent, we subtract from the magic number 0x5f3759df. This does a few things: it preserves the mantissa (the non-exponent part, aka 5 in: $5 \cdot 10^8$), handles odd-even exponents, shifting bits from the exponent into the mantissa, and all sorts of funky stuff.

The following code is the fast inverse square root implementation from Quake III Arena (exact original comment written in Quake III Arena Game).

```
 // CPP program for fast inverse square root.
#include<bits/stdc++.h>
```

```
using namespace std;

// function to find the inverse square root
float inverse_rsqrt( float number )
{
    const float threehalfs = 1.5F;

    float x2 = number * 0.5F;
    float y = number;

    // evil floating point bit level hacking
    long i = * ( long * ) &y;

    // value is pre-assumed
    i = 0x5f3759df - ( i >> 1 );
    y = * ( float * ) &i;

    // 1st iteration
    y = y * ( threehalfs - ( x2 * y * y ) );

    // 2nd iteration, this can be removed
    // y = y * ( threehalfs - ( x2 * y * y ) );

    return y;
}

// driver code
int main(){

    int n = 256;
    float f = inverse_rsqrt(n);
    cout << f << endl;

    return 0;
}
```

Output :

```
0.0623942
```

Source

https://www.geeksforgeeks.org/fast-inverse-square-root/

Chapter 140

Fibbinary Numbers (No consecutive 1s in binary)

Fibbinary Numbers (No consecutive 1s in binary) - GeeksforGeeks

Given N, check if the number is Fibbinary Number or not. Fibbinary numbers are integers whose binary representation contains no consecutive ones.

Examples :

```
Input : 10
Output : YES
Explanation: 1010 is the binary representation
             of 10 which does not contains any
             consecutive 1's.

Input : 11
Output : NO
Explanation: 1011 is the binary representation
             of 11, which contains consecutive
             1's
```

The idea to do this is to right shift the number, till n!=0. For every binary representation of 1, check if the last bit found was 1 or not. Get the last bit of binary representation of the integer by doing a (n&1). If the last bit of the binary representation is 1 and the previous bit before doing a right shift was also one, we encounter consecutive 1's. So we come to the conclusion that it is not a fibonnary number.

Some of the first few Fibonnary numbers are:

```
0, 2, 4, 8, 10, 16, 18, 20.......
```

CPP

```
 // CPP program to check if a number
// is fibinnary number or not
#include <iostream>
using namespace std;

// function to check if binary
// representation of an integer
// has consecutive 1s
bool checkFibinnary(int n)
{
    // stores the previous last bit
    // initially as 0
    int prev_last = 0;

    while (n)
    {
        // if current last bit and
        // previous last bit is 1
        if ((n & 1) && prev_last)
            return false;

        // stores the last bit
        prev_last = n & 1;

        // right shift the number
        n >>= 1;
    }

    return true;
}

// Driver code to check above function
int main()
{
    int n = 10;
    if (checkFibinnary(n))
        cout << "YES";
    else
        cout << "NO";
    return 0;
}
```

Java

```
 // Java program to check if a number
```

```
// is fibinnary number or not
class GFG {

    // function to check if binary
    // representation of an integer
    // has consecutive 1s
    static boolean checkFibinnary(int n)
    {

        // stores the previous last bit
        // initially as 0
        int prev_last = 0;

        while (n != 0)
        {

            // if current last bit and
            // previous last bit is 1
            if ((n & 1) != 0 && prev_last != 0)

                return false;

            // stores the last bit
            prev_last = n & 1;

            // right shift the number
            n >>= 1;
        }

        return true;
    }

    // Driver code to check above function
    public static void main(String[] args)
    {
        int n = 10;

        if (checkFibinnary(n) == true)
            System.out.println("YES");
        else
            System.out.println("NO");
    }
}

// This code is contributed by
// Smitha Dinesh Semwal
```

Python3

```
 # Python 3 program to check if a
# number is fibinnary number or
# not

# function to check if binary
# representation of an integer
# has consecutive 1s
def checkFibinnary(n):

    # stores the previous last bit
    # initially as 0
    prev_last = 0

    while (n):

        # if current last bit and
        # previous last bit is 1
        if ((n & 1) and prev_last):
            return False

        # stores the last bit
        prev_last = n & 1

        # right shift the number
        n >>= 1


    return True

# Driver code
n = 10

if (checkFibinnary(n)):
    print("YES")
else:
    print("NO")

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to check if a number
// is fibinnary number or not
using System;

class GFG {

    // function to check if binary
```

```
    // representation of an integer
    // has consecutive 1s
    static bool checkFibinnary(int n)
    {

        // stores the previous last bit
        // initially as 0
        int prev_last = 0;

        while (n != 0)
        {

            // if current last bit and
            // previous last bit is 1
            if ((n & 1) != 0 && prev_last != 0)

                return false;

            // stores the last bit
            prev_last = n & 1;

            // right shift the number
            n >>= 1;
        }

        return true;
    }

    // Driver code to check above function
    public static void Main()
    {
        int n = 10;

        if (checkFibinnary(n) == true)
            Console.WriteLine("YES");
        else
            Console.WriteLine("NO");
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to check if a number
// is fibinnary number or not
```

```
// function to check if binary
// representation of an integer
// has consecutive 1s
function checkFibinnary($n)
{
    // stores the previous last bit
    // initially as 0
    $prev_last = 0;

    while ($n)
    {
        // if current last bit and
        // previous last bit is 1
        if (($n & 1) && $prev_last)
            return false;

        // stores the last bit
        $prev_last = $n & 1;

        // right shift the number
        $n >>= 1;
    }
    return true;
}

// Driver code
$n = 10;
if (checkFibinnary($n))
    echo "YES";
else
    echo "NO";

// This code is contributed by mits
?>
```

Output :

```
YES
```

Time Complexity: O(log(n))

Fibbinary Numbers (No consecutive 1s in binary) – O(1) Approach

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/httpswww-geeksforgeeks-orgfibbinary-numbers/

Chapter 141

Fibbinary Numbers (No consecutive 1s in binary) – O(1) Approach

Fibbinary Numbers (No consecutive 1s in binary) - O(1) Approach - GeeksforGeeks

Given a positive integer **n**. The problem is to check if the number is Fibbinary Number or not. Fibbinary numbers are integers whose binary representation contains no consecutive ones.

Examples :

```
Input : 10
Output : Yes
Explanation: 1010 is the binary representation
             of 10 which does not contains any
             consecutive 1's.

Input : 11
Output : No
Explanation: 1011 is the binary representation
             of 11, which contains consecutive
             1's.
```

Approach: If (n & (n >> 1)) == 0, then 'n' is a fibbinary number Else not.

C++

```
 // C++ implementation to check whether a number
// is fibbinary or not
```

```
#include <bits/stdc++.h>
using namespace std;

// function to check whether a number
// is fibbinary or not
bool isFibbinaryNum(unsigned int n) {

  // if the number does not contain adjacent ones
  // then (n & (n >> 1)) operation results to 0
  if ((n & (n >> 1)) == 0)
    return true;

  // not a fibbinary number
  return false;
}

// Driver program to test above
int main() {
  unsigned int n = 10;
  if (isFibbinaryNum(n))
    cout << "Yes";
  else
    cout << "No";
  return 0;
}
```

Java

```
 // Java implementation to check whether
// a number is fibbinary or not
class GFG {

    // function to check whether a number
    // is fibbinary or not
    static boolean isFibbinaryNum(int n) {

        // if the number does not contain
        // adjacent ones then (n & (n >> 1))
        // operation results to 0
        if ((n & (n >> 1)) == 0)
            return true;

        // not a fibbinary number
        return false;
    }

    // Driver program to test above
    public static void main(String[] args) {
```

```
        int n = 10;

        if (isFibbinaryNum(n) == true)
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

// This code is contributed by
// Smitha Dinesh Semwal
```

Python3

```
 # Python3 program to check if a number
# is fibinnary number or not

# function to check whether a number
# is fibbinary or not
def isFibbinaryNum( n):

    # if the number does not contain adjacent
    # ones then (n & (n >> 1)) operation
    # results to 0
    if ((n & (n >> 1)) == 0):
        return 1

    # Not a fibbinary number
    return 0

# Driver code
n = 10

if (isFibbinaryNum(n)):
    print("Yes")
else:
    print("No")

# This code is contributed by sunnysingh
```

C#

```
 // C# implementation to check whether
// a number is fibbinary or not
using System;
```

```
class GFG {

    // function to check whether a number
    // is fibbinary or not
    static bool isFibbinaryNum(int n) {

        // if the number does not contain
        // adjacent ones then (n & (n >> 1))
        // operation results to 0
        if ((n & (n >> 1)) == 0)
            return true;

        // not a fibbinary number
        return false;
    }

    // Driver program to test above
    public static void Main() {

        int n = 10;

        if (isFibbinaryNum(n) == true)
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP implementation to check whether
// a number is fibbinary or not

// function to check whether a number
// is fibbinary or not
function isFibbinaryNum($n)
{

    // if the number does not contain
    // adjacent ones then (n & (n >> 1))
    // operation results to 0
    if (($n & ($n >> 1)) == 0)
        return true;

    // not a fibbinary number
```

```
    return false;
}

// Driver code
$n = 10;
if (isFibbinaryNum($n))
    echo "Yes";
else
    echo "No";

// This code is contributed by mits
?>
```

Output :

```
Yes
```

Time Complexity: O(1).

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/fibbinary-numbers-no-consecutive-1s-binary-o1-approach/

Chapter 142

Find Duplicates of array using bit array

Find Duplicates of array using bit array - GeeksforGeeks

You have an array of N numbers, where N is at most 32,000. The array may have duplicates entries and you do not know what N is. With only 4 Kilobytes of memory available, how would print all duplicates elements in the array ?.

Examples:

```
Input : arr[] = {1, 5, 1, 10, 12, 10}
Output : 1 10
1 and 10 appear more than once in given
array.

Input : arr[] = {50, 40, 50}
Output : 50
```

Asked In: Amazon

We have 4 Kilobytes of memory which means we can address up to 8 * 4 * 2^{10} bits. Note that 32 * 2^{10} bits is greater than 32000. We can create a bit with 32000 bits, where each bit represents one integer.

Note: If you need to create a bit with more than 32000 bits then you can create easily more and more than 32000;

Using this bit vector, we can then iterate through the array, flagging each element v by setting bit v to 1. When we come across a duplicate element, we print it.

Below is Java implementation of the idea.

```
 // Java program to print all Duplicates in array
import java.util.*;
import java.lang.*;
import java.io.*;

// A class to represent array of bits using
// array of integers
class BitArray
{
    int[] arr;

    // Constructor
    public BitArray(int n)
    {
        // Devide by 32. To store n bits, we need
        // n/32 + 1 integers (Assuming int is stored
        // using 32 bits)
        arr = new int[(n>>5) + 1];
    }

    // Get value of a bit at given position
    boolean get(int pos)
    {
        // Divide by 32 to find position of
        // integer.
        int index = (pos >> 5);

        // Now find bit number in arr[index]
        int bitNo  = (pos & 0x1F);

        // Find value of given bit number in
        // arr[index]
        return (arr[index] & (1 << bitNo)) != 0;
    }

    // Sets a bit at given position
    void set(int pos)
    {
        // Find index of bit position
        int index = (pos >> 5);

        // Set bit number in arr[index]
        int bitNo = (pos & 0x1F);
        arr[index] |= (1 << bitNo);
    }

    // Main function to print all Duplicates
    static void checkDuplicates(int[] arr)
```

```
    {
        // create a bit with 32000 bits
        BitArray ba = new BitArray(320000);

        // Traverse array elements
        for (int i=0; i<arr.length; i++)
        {
            // Index in bit array
            int num  = arr[i] - 1;

            // If num is already present in bit array
            if (ba.get(num))
                System.out.print(num +" ");

            // Else insert num
            else
                ba.set(num);
        }
    }

    // Driver code
    public static void main(String[] args) throws
                                java.lang.Exception
    {
        int[] arr = {1, 5, 1, 10, 12, 10};
        checkDuplicates(arr);
    }
}
```

Output:

```
1 10
```

Source

https://www.geeksforgeeks.org/find-duplicates-of-array-using-bit-array/

Chapter 143

Find Next Sparse Number

Find Next Sparse Number - GeeksforGeeks

A number is Sparse if there are no two adjacent 1s in its binary representation. For example 5 (binary representation: 101) is sparse, but 6 (binary representation: 110) is not sparse. Given a number x, find the smallest Sparse number which greater than or equal to x.

Examples:

```
Input: x = 6
Output: Next Sparse Number is 8

Input: x = 4
Output: Next Sparse Number is 4

Input: x = 38
Output: Next Sparse Number is 40

Input: x = 44
Output: Next Sparse Number is 64
```

A **Simple Solution** is to do following:

```
1) Write a utility function isSparse(x) that takes a number
   and returns true if x is sparse, else false.  This function
   can be easily written by traversing the bits of input number.
2) Start from x and do following
   while(1)
   {
      if (isSparse(x))
         return x;
```

```
      else
         x++
   }
```

Time complexity of isSparse() is O(Log x). Time complexity of this solution is O(x Log x). The next sparse number can be at most O(x) distance away.

Thanks to **kk_angel** for suggesting above solution.

An **Efficient Solution** can solve this problem without checking all numbers on by one. Below are steps.

```
1) Find binary of the given number and store it in a
   boolean array.
2) Initialize last_finalized bit position as 0.
2) Start traversing the binary from least significant bit.
    a) If we get two adjacent 1's such that next (or third)
       bit is not 1, then
          (i)  Make all bits after this 1 to last finalized
               bit (including last finalized) as 0.
          (ii) Update last finalized bit as next bit.
```

For example, let binary representation be 01010001011101, we change it to 01010001**100000** (all bits after highlighted 11 are set to 0). Again two 1's are adjacent, so change binary representation to 010100**10000000**. This is our final answer.

Below is C++ implementation of above solution.

```
 // C++ program to find next sparse number
#include<bits/stdc++.h>
using namespace std;

int nextSparse(int x)
{
    // Find binary representation of x and store it in bin[].
    // bin[0] contains least significant bit (LSB), next
    // bit is in bin[1], and so on.
    vector<bool> bin;
    while (x != 0)
    {
        bin.push_back(x&1);
        x >>= 1;
    }

    // There my be extra bit in result, so add one extra bit
    bin.push_back(0);
    int n = bin.size();  // Size of binary representation
```

```
    // The position till which all bits are finalized
    int last_final = 0;

    // Start from second bit (next to LSB)
    for (int i=1; i<n-1; i++)
    {
       // If current bit and its previous bit are 1, but next
       // bit is not 1.
       if (bin[i] == 1 && bin[i-1] == 1 && bin[i+1] != 1)
       {
            // Make the next bit 1
            bin[i+1] = 1;

            // Make all bits before current bit as 0 to make
            // sure that we get the smallest next number
            for (int j=i; j>=last_final; j--)
                bin[j] = 0;

            // Store position of the bit set so that this bit
            // and bits before it are not changed next time.
            last_final = i+1;
        }
    }

    // Find decimal equivalent of modified bin[]
    int ans = 0;
    for (int i =0; i<n; i++)
        ans += bin[i]*(1<<i);
    return ans;
}

// Driver program
int main()
{
    int x = 38;
    cout << "Next Sparse Number is " << nextSparse(x);
    return 0;
}
```

Output:

```
Next Sparse Number is 40
```

Time complexity of this solution is O(Log x).

Thanks to **gccode** for suggesting above solution.

Source

https://www.geeksforgeeks.org/given-a-number-find-next-sparse-number/

Chapter 144

Find One's Complement of an Integer

Find One's Complement of an Integer - GeeksforGeeks

Given an integer n, find the one's complement of the integer.

Examples:

```
Input  : n = 5
Output : 2

Input  : n = 255
Output : 0

Input  : n = 26
Output : 5
```

An efficient approach to this problem is as follows:
1. Find the number of bits in the given integer
2. XOR the given integer with 2^number_of_bits-1

C++

```
 // CPP program to find 1's complement of n.
#include<bits/stdc++.h>
using namespace std;

unsigned int onesComplement(unsigned int n)
{
   // Find number of bits in the given integer
   int number_of_bits = floor(log2(n))+1;
```

```
   // XOR the given integer with poe(2,
   // number_of_bits-1 and print the result
   return ((1 << number_of_bits) - 1) ^ n;
}

int main()
{
  unsigned int n = 22;
  cout << onesComplement(n);
  return 0;
}
```

Java

```
 // Java program to find 1's complement of n.
class GFG {

    static int onesComplement(int n)
    {

        // Find number of bits in the
        // given integer
        int number_of_bits =
               (int)(Math.floor(Math.log(n) /
                           Math.log(2))) + 1;

        // XOR the given integer with poe(2,
        // number_of_bits-1 and print the result
        return ((1 << number_of_bits) - 1) ^ n;
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 22;

        System.out.print(onesComplement(n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find
# 1's complement of n.
```

```
import math

def onesComplement(n):

    # Find number of bits in
    # the given integer
    number_of_bits = (int)(math.floor(math.log(n) /
                                math.log(2))) + 1;

    # XOR the given integer with poe(2,
    # number_of_bits-1 and print the result
    return ((1 << number_of_bits) - 1) ^ n;

# Driver code
n = 22
print(onesComplement(n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to find 1's complement of n.
using System;

class GFG {

    static int onesComplement(int n)
    {

       // Find number of bits in the given integer
       int number_of_bits = (int)(Math.Floor(
                   Math.Log(n) / Math.Log(2))) + 1;

       // XOR the given integer with poe(2,
       // number_of_bits-1 and print the result
       return ((1 << number_of_bits) - 1) ^ n;
    }

    //Driver code
    public static void Main ()
    {

        int n = 22;

        Console.WriteLine(onesComplement(n));
    }
}
```

```
// This code is contributed by Anant Agarwal.
```

Output :

```
9
```

Source

https://www.geeksforgeeks.org/find-ones-complement-integer/

Chapter 145

Find Two Missing Numbers | Set 2 (XOR based solution)

Find Two Missing Numbers | Set 2 (XOR based solution) - GeeksforGeeks

Given an array of n unique integers where each element in the array is in range [1, n]. The array has all distinct elements and size of array is (n-2). Hence Two numbers from the range are missing from this array. Find the two missing numbers.

Examples:

```
Input  : arr[] = {1, 3, 5, 6}, n = 6
Output : 2 4

Input : arr[] = {1, 2, 4}, n = 5
Output : 3 5

Input : arr[] = {1, 2}, n = 4
Output : 3 4
```

Find Two Missing Numbers | Set 1 (An Interesting Linear Time Solution)

We have discussed two methods to solve this problem in above article. The method 1 requires O(n) extra space and method 2 can causes overflow. In this post, a new solution is discussed. The solution discussed here is O(n) time, O(1) extra space and causes no overflow.

Below are steps.

1. Find XOR of all array elements and natural numbers from 1 to n. Let the array be arr[] = {1, 3, 5, 6}

```
XOR = (1 ^ 3  ^ 5 ^ 6) ^ (1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6)
```

2. As per the property of XOR, same elements will cancel out and we will be left with 2 XOR 4 = 6 (110). But we don't know the exact numbers,let them be X and Y.
3. A bit is set in xor only if corresponding bits in X and Y are different. This is the crucial step to understand.
4. We take a set bit in XOR. Let us consider the rightmost set bit in XOR, set_bit_no = 010
5. Now again if we XOR all the elements of arr[] and 1 to n that have rightmost bit set we will get one of the repeating numbers, say x.

   ```
   Ex: Elements in arr[] with bit set: {3, 6}
   Elements from 1 to n with bit set {2, 3, 6}
   Result of XOR'ing all these is x = 2.
   ```

6. Similarly, if we XOR all the elements of arr[] and 1 to n that have rightmost bit not set, we will get the other element, say y.

   ```
   Ex: Elements in arr[] with bit not set: {1, 5}
   Elements from 1 to n with bit not set {1, 4, 5}
   Result of XOR'ing all these is y = 4
   ```

Below is the implementation of above steps.

C++

```
 // C++ Program to find 2 Missing Numbers using O(1)
// extra space and no overflow.
#include<bits/stdc++.h>

// Function to find two missing numbers in range
// [1, n]. This function assumes that size of array
// is n-2 and all array elements are distinct
void findTwoMissingNumbers(int arr[], int n)
{
    /* Get the XOR of all elements in arr[] and
       {1, 2 .. n} */
    int XOR = arr[0];
    for (int i = 1; i < n-2; i++)
        XOR ^= arr[i];
    for (int i = 1; i <= n; i++)
        XOR ^= i;

    // Now XOR has XOR of two missing elements. Any set
    // bit in it must be set in one missing and unset in
```

```
    // other missing number

    // Get a set bit of XOR (We get the rightmost set bit)
    int set_bit_no = XOR & ~(XOR-1);

    // Now divide elements in two sets by comparing rightmost
    // set bit of XOR with bit at same position in each element.
    int x = 0, y = 0; // Initialize missing numbers
    for (int i = 0; i < n-2; i++)
    {
        if (arr[i] & set_bit_no)
            x = x ^ arr[i]; /*XOR of first set in arr[] */
        else
            y = y ^ arr[i]; /*XOR of second set in arr[] */
    }
    for (int i = 1; i <= n; i++)
    {
        if (i & set_bit_no)
            x = x ^ i; /* XOR of first set in arr[] and
                          {1, 2, ...n }*/
        else
            y = y ^ i; /* XOR of second set in arr[] and
                          {1, 2, ...n } */
    }

    printf("Two Missing Numbers are\n %d %d", x, y);
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 3, 5, 6};

    // Range of numbers is 2 plus size of array
    int n = 2 + sizeof(arr)/sizeof(arr[0]);

    findTwoMissingNumbers(arr, n);

    return 0;
}
```

Java

```
 // Java Program to find 2 Missing Numbers
import java.util.*;

class GFG {
```

```
// Function to find two missing numbers in range
// [1, n]. This function assumes that size of array
// is n-2 and all array elements are distinct
static void findTwoMissingNumbers(int arr[], int n)
{
    /* Get the XOR of all elements in arr[] and
       {1, 2 .. n} */
    int XOR = arr[0];
    for (int i = 1; i < n-2; i++)
        XOR ^= arr[i];
    for (int i = 1; i <= n; i++)
        XOR ^= i;

    // Now XOR has XOR of two missing elements.
    // Any set bit in it must be set in one missing
    // and unset in other missing number

    // Get a set bit of XOR (We get the rightmost
    // set bit)
    int set_bit_no = XOR & ~(XOR-1);

    // Now divide elements in two sets by comparing
    // rightmost set bit of XOR with bit at same
    // position in each element.
    int x = 0, y = 0; // Initialize missing numbers
    for (int i = 0; i < n-2; i++)
    {
        if ((arr[i] & set_bit_no) > 0)

            /*XOR of first set in arr[] */
            x = x ^ arr[i];
        else
            /*XOR of second set in arr[] */
            y = y ^ arr[i];
    }

    for (int i = 1; i <= n; i++)
    {
        if ((i & set_bit_no)>0)

            /* XOR of first set in arr[] and
               {1, 2, ...n }*/
            x = x ^ i;
        else
            /* XOR of second set in arr[] and
                {1, 2, ...n } */
            y = y ^ i;
    }
```

```
        System.out.println("Two Missing Numbers are ");
        System.out.println( x + " " + y);
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
         int arr[] = {1, 3, 5, 6};

         // Range of numbers is 2 plus size of array
         int n = 2 +arr.length;

         findTwoMissingNumbers(arr, n);

        }
    }

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python Program to find 2 Missing
# Numbers using O(1)
# extra space and no overflow.

# Function to find two missing
# numbers in range
# [1, n]. This function assumes
# that size of array
# is n-2 and all array elements
# are distinct
def findTwoMissingNumbers(arr, n):

        # Get the XOR of all
        # elements in arr[] and
    # {1, 2 .. n}
    XOR = arr[0]
    for i in range(1,n-2):
        XOR ^= arr[i]
    for i in range(1,n+1):
        XOR ^= i

    # Now XOR has XOR of two
        # missing elements. Any set
    # bit in it must be set in
        # one missing and unset in
    # other missing number
```

```
    # Get a set bit of XOR
        # (We get the rightmost set bit)
    set_bit_no = XOR & ~(XOR-1)

    # Now divide elements in two sets
        # by comparing rightmost
    # set bit of XOR with bit at same
        # position in each element.
    x = 0

        # Initialize missing numbers
    y = 0
    for i in range(0,n-2):
        if arr[i] & set_bit_no:

                # XOR of first set in arr[]
            x = x ^ arr[i]
        else:

                # XOR of second set in arr[]
            y = y ^ arr[i]
    for i in range(1,n+1):
        if i & set_bit_no:

                # XOR of first set in arr[] and
                # {1, 2, ...n }
            x = x ^ i

        else:

                # XOR of second set in arr[] and
                # {1, 2, ...n }
            y = y ^ i


    print ("Two Missing Numbers are\n%d %d"%(x,y))

# Driver program to test
# above function
arr = [1, 3, 5, 6]

# Range of numbers is 2
# plus size of array
n = 2 + len(arr)
findTwoMissingNumbers(arr, n)

# This code is contributed
```

```
# by Shreyanshi Arun.
```

C#

```
 // Program to find 2 Missing Numbers
using System;

class GFG {

    // Function to find two missing
    // numbers in range [1, n].This
    // function assumes that size of
    // array is n-2 and all array
    // elements are distinct
    static void findTwoMissingNumbers(int[] arr, int n)
    {
        // Get the XOR of all elements
        // in arr[] and {1, 2 .. n}
        int XOR = arr[0];

        for (int i = 1; i < n - 2; i++)
            XOR ^= arr[i];

        for (int i = 1; i <= n; i++)
            XOR ^= i;

        // Now XOR has XOR of two missing
        // element. Any set bit in it must
        // be set in one missing and unset
        // in other missing number
        // Get a set bit of XOR (We get the
        // rightmost set bit)
        int set_bit_no = XOR & ~(XOR - 1);

        // Now divide elements in two sets
        // by comparing rightmost set bit
        // of XOR with bit at same position
        // in each element.
        int x = 0, y = 0;

        // Initialize missing numbers
        for (int i = 0; i < n - 2; i++) {

            if ((arr[i] & set_bit_no) > 0)

                // XOR of first set in arr[]
                x = x ^ arr[i];
```

```
            else
                // XOR of second set in arr[]
                y = y ^ arr[i];
        }

        for (int i = 1; i <= n; i++) {
            if ((i & set_bit_no) > 0)
                // XOR of first set in arr[]
                // and {1, 2, ...n }
                x = x ^ i;

            else
                // XOR of second set in arr[]
                // and {1, 2, ...n }
                y = y ^ i;
        }

        Console.WriteLine("Two Missing Numbers are ");
        Console.WriteLine(x + " " + y);
    }

    // Driver program
    public static void Main()
    {
        int[] arr = { 1, 3, 5, 6 };

        // Range of numbers is 2 plus
        // size of array
        int n = 2 + arr.Length;

        findTwoMissingNumbers(arr, n);
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP Program to find 2 Missing
// Numbers using O(1) extra
// space and no overflow.

// Function to find two
// missing numbers in range
// [1, n]. This function
// assumes that size of array
// is n-2 and all array
```

```
// elements are distinct
function findTwoMissingNumbers($arr, $n)
{

    // Get the XOR of all
    // elements in arr[] and
    // {1, 2 .. n}
    $XOR = $arr[0];
    for ($i = 1; $i < $n - 2; $i++)
        $XOR ^= $arr[$i];
    for ($i = 1; $i <= $n; $i++)
        $XOR ^= $i;

    // Now XOR has XOR of two
    // missing elements. Any set
    // bit in it must be set in
    // one missing and unset in
    // other missing number

    // Get a set bit of XOR
    // (We get the rightmost
    // set bit)
    $set_bit_no = $XOR & ~($XOR - 1);

    // Now divide elements in two
    // sets by comparing rightmost
    // set bit of XOR with bit at
    // same position in each element.

    $x = 0;

    // Initialize missing numbers
    $y = 0;
    for ($i = 0; $i < $n - 2; $i++)
    {
        if ($arr[$i] & $set_bit_no)

            // XOR of first set in arr[]
            $x = $x ^ $arr[$i];

        else

            // XOR of second set in arr[]
            $y = $y ^ $arr[$i];
    }
    for ($i = 1; $i <= $n; $i++)
    {
        if ($i & $set_bit_no)
```

```
            // XOR of first set in arr[]
            // and {1, 2, ...n }
            $x = $x ^ $i;

        else

            // XOR of second set in arr[]
            // and {1, 2, ...n }
            $y = $y ^ $i;
    }

    echo "Two Missing Numbers are\n", $x;
    echo "\n", $y;
}

    // Driver Code
    $arr = array(1, 3, 5, 6);

    // Range of numbers is 2
    // plus size of array
    $n = 2 + count($arr);

    findTwoMissingNumbers($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
Two Missing Numbers are
2 4
```

Time Complexity : O(n)
Auxiliary Space : O(1)
No integer overflow

Improved By : vt_m

Source

https://www.geeksforgeeks.org/find-two-missing-numbers-set-2-xor-based-solution/

Chapter 146

Find Unique pair in an array with pairs of numbers

Find Unique pair in an array with pairs of numbers - GeeksforGeeks

Given an array where every element appears twice except a pair (two elements). Find the elements of this unique pair.

Examples:

```
Input  : 6, 1, 3, 5, 1, 3, 7, 6
Output : 5 7
All elements appear twice except 5 and 7

Input  : 1 3 4 1
Output : 3 4
```

The idea is based on below post.

Find Two Missing Numbers | Set 2 (XOR based solution)

1. XOR each element of the array and you will left with the XOR of two different elements which are going to be our result. Let this XOR be "**XOR**"
2. Now find a set bit in **XOR**.
3. Now divide array elements in two groups. One group that has the bit found in step 2 as set and other group that has the bit as 0.
4. XOR of elements present in first group would be our first element. And XOR of elements present in second group would be our second element.

C++

```
// C program to find a unique pair in an array
```

```
// of pairs.
#include <stdio.h>

void findUniquePair(int arr[], int n)
{
    // XOR each element and get XOR of two unique
    // elements(ans)
    int XOR = arr[0];
    for (int i = 1; i < n; i++)
        XOR = XOR ^ arr[i];

    // Now XOR has XOR of two missing elements. Any set
    // bit in it must be set in one missing and unset in
    // other missing number

    // Get a set bit of XOR (We get the rightmost set bit)
    int set_bit_no = XOR & ~(XOR-1);

    // Now divide elements in two sets by comparing rightmost
    // set bit of XOR with bit at same position in each element.
    int x = 0, y = 0; // Initialize missing numbers
    for (int i = 0; i < n; i++)
    {
        if (arr[i] & set_bit_no)
            x = x ^ arr[i]; /*XOR of first set in arr[] */
        else
            y = y ^ arr[i]; /*XOR of second set in arr[] */
    }

    printf("The unique pair is (%d, %d)", x, y);

}

// Driver code
int main()
{
    int a[] = { 6, 1, 3, 5, 1, 3, 7, 6 };
    int n = sizeof(a)/sizeof(a[0]);
    findUniquePair(a, n);
    return 0;
}
```

Java

```
 // Java program to find a unique pair
// in an array of pairs.
class GFG
{
```

```
    static void findUniquePair(int[] arr, int n)
    {
        // XOR each element and get XOR of two
        // unique elements(ans)
        int XOR = arr[0];

        for (int i = 1; i < n; i++)
            XOR = XOR ^ arr[i];

        // Now XOR has XOR of two missing elements.
        // Any set bit in it must be set in one
        // missing and unset in other missing number

        // Get a set bit of XOR (We get the
        // rightmost set bit)
        int set_bit_no = XOR & ~(XOR-1);

        // Now divide elements in two sets by
        // comparing rightmost set bit of XOR with
        // bit at same position in each element.
        // Initialize missing numbers
        int x = 0, y = 0;

        for (int i = 0; i < n; i++)
        {
            if ((arr[i] & set_bit_no)>0)

                /*XOR of first set in arr[] */
                x = x ^ arr[i];
            else
                /*XOR of second set in arr[] */
                y = y ^ arr[i];
        }

        System.out.println("The unique pair is (" +
                                x + "," + y + ")");

    }

    // Driver code
    public static void main (String[] args) {
    int[] a = { 6, 1, 3, 5, 1, 3, 7, 6 };
    int n = a.length;
    findUniquePair(a, n);
    }

}
```

```
/* This code is contributed by Mr. Somesh Awasthi */
```

Python 3

```
 # Pyhton 3 program to find a unique
# pair in an array of pairs.
def findUniquePair(arr, n):

    # XOR each element and get XOR
    # of two unique elements(ans)
    XOR = arr[0]
    for i in range(1, n):
        XOR = XOR ^ arr[i]

    # Now XOR has XOR of two missing
    # elements. Any set bit in it
    # must be set in one missing and
    # unset in other missing number

    # Get a set bit of XOR (We get
    # the rightmost set bit)
    set_bit_no = XOR & ~(XOR - 1)

    # Now divide elements in two sets
    # by comparing rightmost set bit
    # of XOR with bit at same position
    # in each element.
    x = 0
    y = 0 # Initialize missing numbers
    for i in range(0, n):

        if (arr[i] & set_bit_no):

            # XOR of first set in
            # arr[]
            x = x ^ arr[i]
        else:

            # XOR of second set
            # in arr[]
            y = y ^ arr[i]


    print("The unique pair is (", x,
             ", ", y, ")", sep = "")

# Driver code
a = [6, 1, 3, 5, 1, 3, 7, 6 ]
```

```
n = len(a)
findUniquePair(a, n)

# This code is contributed by Smitha.
```

C#

```
 // C# program to find a unique pair
// in an array of pairs.
using System;

class GFG {

    static void findUniquePair(int[] arr, int n)
    {

        // XOR each element and get XOR of two
        // unique elements(ans)
        int XOR = arr[0];

        for (int i = 1; i < n; i++)
            XOR = XOR ^ arr[i];

        // Now XOR has XOR of two missing
        // elements. Any set bit in it must
        // be set in one missing and unset
        // in other missing number

        // Get a set bit of XOR (We get the
        // rightmost set bit)
        int set_bit_no = XOR & ~(XOR - 1);

        // Now divide elements in two sets by
        // comparing rightmost set bit of XOR
        // with bit at same position in each
        // element. Initialize missing numbers
        int x = 0, y = 0;

        for (int i = 0; i < n; i++)
        {
            if ((arr[i] & set_bit_no) > 0)

                /*XOR of first set in arr[] */
                x = x ^ arr[i];
            else

                /*XOR of second set in arr[] */
                y = y ^ arr[i];
```

```
        }

        Console.WriteLine("The unique pair is ("
                          + x + ", " + y + ")");
    }

    // Driver code
    public static void Main ()
    {
        int[] a = { 6, 1, 3, 5, 1, 3, 7, 6 };
        int n = a.Length;

        findUniquePair(a, n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find a
// unique pair in an array
// of pairs.

function findUniquePair($arr, $n)
{

    // XOR each element and
    // get XOR of two unique
    // elements(ans)
    $XOR = $arr[0];
    for ($i = 1; $i < $n; $i++)
        $XOR = $XOR ^ $arr[$i];

    // Now XOR has XOR of two
    // missing elements. Any set
    // bit in it must be set in
    // one missing and unset in
    // other missing number

    // Get a set bit of XOR
    // (We get the rightmost set bit)
    $set_bit_no = $XOR & ~($XOR-1);

    // Now divide elements in two
    // sets by comparing rightmost
    // set bit of XOR with bit at
```

```
    // same position in each element.
    // Initialize missing numbers
    $x = 0;
    $y = 0;
    for ($i = 0; $i < $n; $i++)
    {
        if ($arr[$i] & $set_bit_no)

            // XOR of first set in arr[]
            $x = $x ^ $arr[$i];
        else

            // XOR of second set in arr[]
            $y = $y ^ $arr[$i];
    }

    echo"The unique pair is ", "(",$x," ", $y,")";

}

    // Driver code
    $a = array(6, 1, 3, 5, 1, 3, 7, 6);
    $n = count($a);
    findUniquePair($a, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
The unique pair is (7, 5)
```

Improved By : vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/find-unique-pair-array-pairs-numbers/

Chapter 147

Find XOR of two number without using XOR operator

Find XOR of two number without using XOR operator - GeeksforGeeks

Given two integers, find XOR of them without using XOR operator, i.e., without using ^ in C/C++.

Examples :

```
Input:  x = 1, y = 2
Output: 3

Input:  x = 3, y = 5
Output: 6
```

A **Simple Solution** is to traverse all bits one by one. For every pair of bits, check if both are same, set the corresponding bit as 0 in output, otherwise set as 1.

C++

```
 // C++ program to find XOR without using ^
#include <iostream>
using namespace std;

// Returns XOR of x and y
int myXOR(int x, int y)
{
    int res = 0; // Initialize result

    // Assuming 32-bit Integer
    for (int i = 31; i >= 0; i--)
```

```
    {
       // Find current bits in x and y
       bool b1 = x & (1 << i);
       bool b2 = y & (1 << i);

        // If both are 1 then 0 else xor is same as OR
        bool xoredBit = (b1 & b2) ? 0 : (b1 | b2);

        // Update result
        res <<= 1;
        res |= xoredBit;
    }
    return res;
}

// Driver program to test above function
int main()
{
   int x = 3, y = 5;
   cout << "XOR is " << myXOR(x, y);
   return 0;
}
```

Output :

```
XOR is 6
```

Thanks to Utkarsh Trivedi for suggesting this solution.

A **Better Solution** can find XOR without using loop.

1) Find bitwise OR of x and y (Result has set bits where either x has set or y has set bit). OR of x = 3 (011) and y = 5 (101) is 7 (111)

2) To remove extra set bits find places where both x and y have set bits. The value of expression "~x | ~y" has 0 bits wherever x and y both have set bits.

3) bitwise AND of "(x | y)" and "~x | ~y" produces the required result.

Below is implementation.

C++

```
 // C++ program to find XOR without using ^
#include <iostream>
using namespace std;

// Returns XOR of x and y
int myXOR(int x, int y)
```

```
{
   return (x | y) & (~x | ~y);
}

// Driver program to test above function
int main()
{
   int x = 3, y = 5;
   cout << "XOR is " << myXOR(x, y);
   return 0;
}
```

Java

```
 // Java program to find
// XOR without using ^
import java.io.*;

class GFG
{

// Returns XOR of x and y
static int myXOR(int x, int y)
{
    return (x | y) &
           (~x | ~y);
}

// Driver Code
public static void main (String[] args)
{
    int x = 3, y = 5;
    System.out.println("XOR is "+
                       (myXOR(x, y)));
}
}

// This code is contibuted by ajit
```

Python 3

```
 # Python 3 program to
# find XOR without using ^

# Returns XOR of x and y
def myXOR(x, y):
    return ((x | y) &
```

```
            (~x | ~y))

# Driver Code
x = 3
y = 5
print("XOR is" ,
       myXOR(x, y))

# This code is contributed
# by Smitha
```

C#

```
 // C# program to find
// XOR without using ^
using System;

class GFG
{

// Returns XOR of x and y
static int myXOR(int x, int y)
{
    return (x | y) &
           (~x | ~y);
}

// Driver Code
static public void Main ()
{
    int x = 3, y = 5;
    Console.WriteLine("XOR is "+
                      (myXOR(x, y)));
}
}

// This code is contibuted by m_kit
```

PHP

```
 <?php
// PHP program to find
// XOR without using ^

// Returns XOR of x and y
function myXOR($x, $y)
{
```

```
    return ($x | $y) & (~$x | ~$y);
}

// Driver Code
$x = 3;
$y = 5;

echo "XOR is " , myXOR($x, $y);

// This code is contibuted by aj_36
?>
```

Output :

```
XOR is 6
```

Thanks to **jitu_the_best** for suggesting this solution.

Improved By : jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/find-xor-of-two-number-without-using-xor-operator/

Chapter 148

Find even occurring elements in an array of limited range

Find even occurring elements in an array of limited range - GeeksforGeeks

Given an array that contains odd number of occurrences for all numbers except for a few elements which are present even number of times. Find the elements which have even occurrences in the array in O(n) time complexity and O(1) extra space.
Assume array contain elements in the range 0 to 63.

Examples :

```
Input: [9, 12, 23, 10, 12, 12, 15, 23, 14, 12, 15]
Output: 12, 23 and 15

Input: [1, 4, 7, 5, 9, 7, 3, 4, 6, 8, 3, 0, 3]
Output: 4 and 7

Input: [4, 4, 10, 10, 4, 4, 4, 4, 10, 10]
Output: 4 and 10
```

A simple method would be to traverse the array and store frequencies of its elements in a map. Later, print elements that have even count in the map. The solution takes O(n) time but requires extra space for storing frequencies. Below is an interesting method to solve this problem using bitwise operators.

This method assumes that long long integers are stored using 64 bits. The idea is to use XOR operator. We know that

```
1 XOR 1 = 0
```

```
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0
```

Consider below input –

```
 1, 4, 7, 5, 9, 7, 3, 4, 6, 8, 3, 0, 3
```

If we right shift 1 by value of each element of the array and take XOR of all the items, we will get below binary integer –

```
1101101011
```

Each 1 in the i'th index from the right represents odd occurrence of element i. And each 0 in the i'th index from the right represents even or non-occurrence of element i in the array.

0 is present in 2nd, 4th and 7th position in above binary number. But 2 is not present in our array. So our answer is 4 and 7.

Below is C++ implementation of above idea

C++

```
 // C++ Program to find the even occurring elements
// in given array
#include <iostream>
using namespace std;

// Function to find the even occurring elements
// in given array
void printRepeatingEven(int arr[], int n)
{
    long long _xor = 0L;
    long long pos;

    // do for each element of array
    for( int i = 0; i < n; ++i)
    {
        // right pos 1 by value of current element
        pos = 1 << arr[i];

        // Toggle the bit everytime element gets repeated
        _xor ^= pos;
    }

    // Traverse array again and use _xor to find even
    // occuring elements
    for (int i = 0; i < n; ++i)
```

```
    {
        // right shift 1 by value of current element
        pos = 1 << arr[i];

        // Each 0 in _xor represents an even occurrence
        if (!(pos & _xor))
        {
            // print the even occurring numbers
            cout << arr[i] << " ";

            // set bit as 1 to avoid printing duplicates
            _xor ^= pos;
        }
    }
}

// Driver code
int main()
{
    int arr[] = { 9, 12, 23, 10, 12, 12, 15, 23,
                  14, 12, 15 };
    int n = sizeof(arr) / sizeof(arr[0]);

    printRepeatingEven(arr, n);

    return 0;
}
```

PHP

```
 <?php
// PHP Program to find the even
// occurring elements in given array

// Function to find the even
// occurring elements in given array
function printRepeatingEven($arr, $n)
{
    $_xor = 0;
    $pos;

    // do for each element of array
    for( $i = 0; $i < $n; ++$i)
    {
        // right pos 1 by value
        // of current element
        $pos = 1 << $arr[$i];
```

```
        // Toggle the bit everytime
        // element gets repeated
        $_xor ^= $pos;
    }

    // Traverse array again and
    // use _xor to find even
    // occuring elements
    for ($i = 0; $i < $n; ++$i)
    {
        // right shift 1 by value
        // of current element
        $pos = 1 << $arr[$i];

        // Each 0 in _xor represents
        // an even occurrence
        if (!($pos & $_xor))
        {
            // print the even
            // occurring numbers
            echo $arr[$i], " ";

            // set bit as 1 to avoid
            // printing duplicates
            $_xor ^= $pos;
        }
    }
}

// Driver code
$arr = array(9, 12, 23, 10, 12, 12,
             15, 23, 14, 12, 15 );
$n = sizeof($arr);

printRepeatingEven($arr, $n);

// This code is contributed by aj_36
?>
```

Output :

```
12 23 15
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/find-even-occurring-elements-array-limited-range/

Chapter 149

Find i'th index character in a binary string obtained after n iterations | Set 2

Find i'th index character in a binary string obtained after n iterations | Set 2 - GeeksforGeeks

Given a decimal number m, convert it into a binary string and apply n iterations, in each iteration 0 becomes "01" and 1 becomes "10". Find ith(based indexing) index character in the string after nth iteration.

Examples:

```
Input: m = 5 i = 5 n = 3
Output: 1
Explanation
In the first case m = 5, i = 5, n = 3.
Initially, the string is  101  ( binary equivalent of 5 )
After 1st iteration -   100110
After 2nd iteration - 100101101001
After 3rd iteration -   100101100110100110010110
The character at index 5 is 1, so 1 is the answer

Input: m = 11 i = 6 n = 4
Output: 1
```

A **naive approach** to this problem has been discussed in the previous post.

Efficient algorithm: The first step will be to find which block the i-th character will be after N iterations are performed. In the n'th iteration distance between any two consecutive characters initially will always be equal to 2^n. For a general number m, the number of blocks will be ceil(log m). If M was 3, the string gets divided into 3 blocks. Find the block number in which kth character will lie by k / (2^n), where n is the number of iterations.

Consider m=5, then the binary representation is 101. Then the distance between any 2 consecutive marked characters in any i'th iteration will be as follows

0th iteration: 101, distance = 0
1st iteration: **1**0 **0**1 **1** 0, distance = 2
2nd iteration: 1001 **0**110 **1**001, distance = 4
3rd iteration: **1**0010110 **0**1101001 **1**0010110, distance = 8

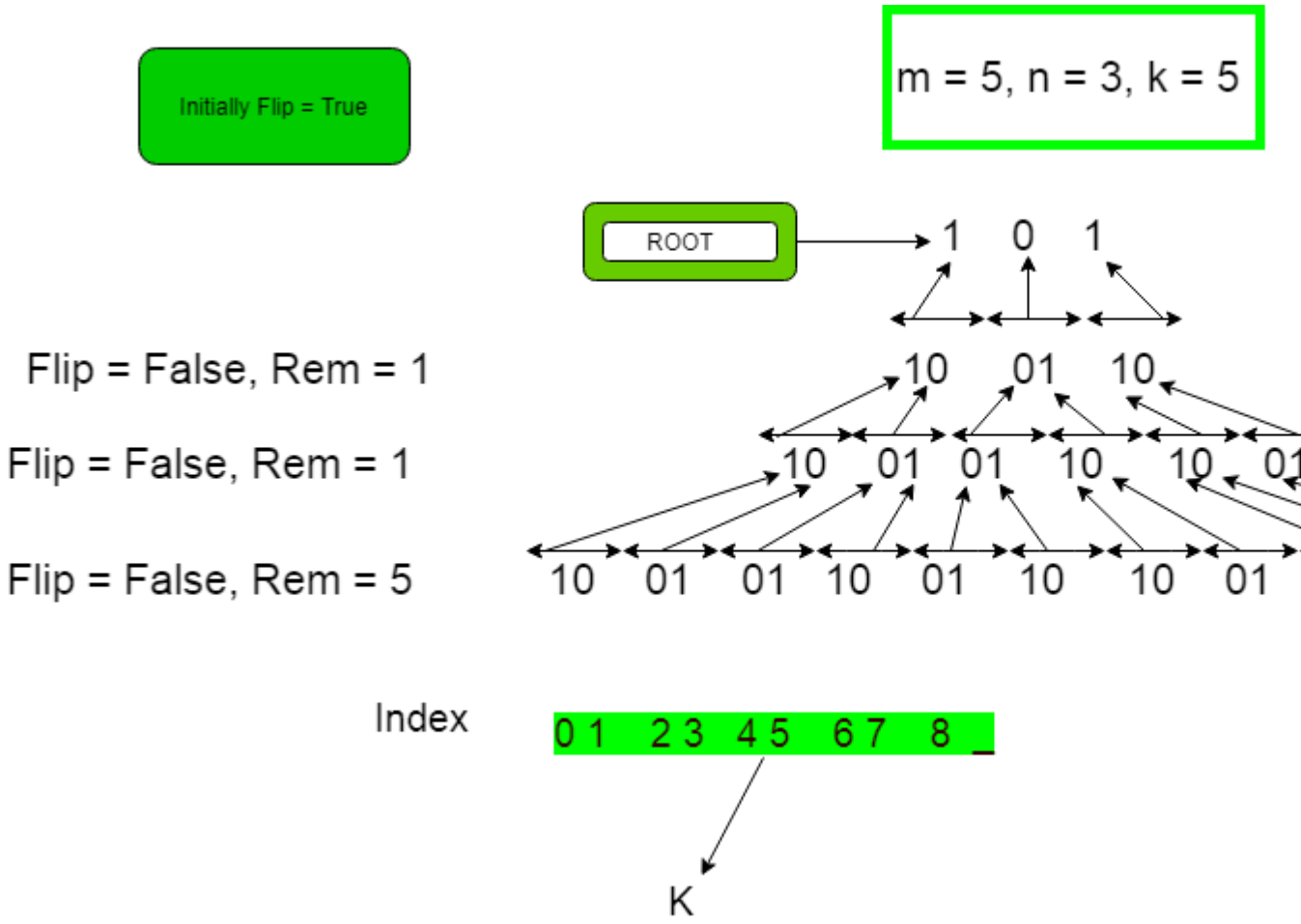
In the example k = 5 and n = 3, so Block_number, when k is 5, will be 0, as 5 / (2^3) = 0

Initially, block numbers will be

```
Original String :    1   0    1
Block_number    :    0   1    2
```

There is no need to generate the entire string, only computing in the block in which the i-th character is present will give the answer. Let this character be root **root = s[Block_number]**, where s is the binary representation of "m". Now in the final string, find the distance of the kth character from the block number, call this distance as remaining. So **remaining = k % (2^n)** will be the index of i-th character in the block. If remaining is 0, the root will be the answer. Now, in order to check whether the root is the actual answer use a boolean variable *flip* which whether we need to flip our answer or not. Following the below algorithm will give the character at the i-th index.

```
bool flip = true;
while(remaining > 1){
   if( remaining is odd )
        flip = !flip
   remaining = remaining/2;
}
```



Below is the implementation of the above approach:

C++

```
 // C++ program to find i'th Index character
// in a binary string obtained after n iterations
#include <bits/stdc++.h>
using namespace std;

// Function to find the i-th character
void KthCharacter(int m, int n, int k)
{
    // distance between two consecutive
    // elements after N iterations
    int distance = pow(2, n);
    int Block_number = k / distance;
    int remaining = k % distance;
```

```
    int s[32], x = 0;

    // binary representation of M
    for (; m > 0; x++) {
        s[x] = m % 2;
        m = m / 2;
    }

    // kth digit will be derived from root for sure
    int root = s[x - 1 - Block_number];

    if (remaining == 0) {
        cout << root << endl;
        return;
    }

    // Check whether there is need to
    // flip root or not
    bool flip = true;
    while (remaining > 1) {
        if (remaining & 1) {
            flip = !flip;
        }
        remaining = remaining >> 1;
    }

    if (flip) {
        cout << !root << endl;
    }
    else {
        cout << root << endl;
    }
}

// Driver Code
int main()
{
    int m = 5, k = 5, n = 3;
    KthCharacter(m, n, k);
    return 0;
}
```

Java

```
 // Java program to find ith
// Index character in a binary
// string obtained after n iterations
```

```
import java.io.*;

class GFG
{
// Function to find
// the i-th character
static void KthCharacter(int m,
                         int n, int k)
{
    // distance between two
    // consecutive elements
    // after N iterations
    int distance = (int)Math.pow(2, n);
    int Block_number = k / distance;
    int remaining = k % distance;

    int s[] = new int[32];
    int x = 0;

    // binary representation of M
    for (; m > 0; x++)
    {
        s[x] = m % 2;
        m = m / 2;
    }

    // kth digit will be
    // derived from root
    // for sure
    int root = s[x - 1 -
                 Block_number];

    if (remaining == 0)
    {
        System.out.println(root);
        return;
    }

    // Check whether there is
    // need to flip root or not
    Boolean flip = true;
    while (remaining > 1)
    {
        if ((remaining & 1) > 0)
        {
            flip = !flip;
        }
        remaining = remaining >> 1;
```

```
    }

    if (flip)
    {
        System.out.println(!(root > 0));
    }
    else
    {
        System.out.println(root);
    }
}

// Driver Code
public static void main (String[] args)
{
    int m = 5, k = 5, n = 3;
    KthCharacter(m, n, k);
}
}

// This code is contributed
// by anuj_67.
```

Python3

```
 # Python3 program to find
# i'th Index character in
# a binary string obtained
# after n iterations

# Function to find
# the i-th character
def KthCharacter(m, n, k):

    # distance between two
    # consecutive elements
    # after N iterations
    distance = pow(2, n)
    Block_number = int(k / distance)
    remaining = k % distance

    s = [0] * 32
    x = 0

    # binary representation of M
    while(m > 0) :
        s[x] = m % 2
        m = int(m / 2)
```

```
        x += 1

    # kth digit will be derived
    # from root for sure
    root = s[x - 1 - Block_number]

    if (remaining == 0):
        print(root)
        return

    # Check whether there
    # is need to flip root
    # or not
    flip = True
    while (remaining > 1):
        if (remaining & 1):
            flip = not(flip)

        remaining = remaining >> 1

    if (flip) :
        print(not(root))

    else :
        print(root)

# Driver Code
m = 5
k = 5
n = 3
KthCharacter(m, n, k)

# This code is contributed
# by smita
```

C#

```
 // C# program to find ith
// Index character in a
// binary string obtained
// after n iterations
using System;

class GFG
{
// Function to find
// the i-th character
static void KthCharacter(int m,
```

```
                        int n,
                        int k)
{
    // distance between two
    // consecutive elements
    // after N iterations
    int distance = (int)Math.Pow(2, n);
    int Block_number = k / distance;
    int remaining = k % distance;

    int []s = new int[32];
    int x = 0;

    // binary representation of M
    for (; m > 0; x++)
    {
        s[x] = m % 2;
        m = m / 2;
    }

    // kth digit will be
    // derived from root
    // for sure
    int root = s[x - 1 -
                Block_number];

    if (remaining == 0)
    {
        Console.WriteLine(root);
        return;
    }

    // Check whether there is
    // need to flip root or not
    Boolean flip = true;
    while (remaining > 1)
    {
        if ((remaining & 1) > 0)
        {
            flip = !flip;
        }

        remaining = remaining >> 1;
    }

    if (flip)
    {
        Console.WriteLine(!(root > 0));
```

```
    }
    else
    {
        Console.WriteLine(root);
    }
}

// Driver Code
public static void Main ()
{
    int m = 5, k = 5, n = 3;
    KthCharacter(m, n, k);
}
}

// This code is contributed
// by anuj_67.
```

Output:

```
1
```

Time Complexity: O(log Z), where Z is the distance between initially consecutive bits after N iterations

Improved By : vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/find-ith-index-character-in-a-binary-string-obtained-after-n-iterations-set-2/

Chapter 150

Find largest element from array without using conditional operator

Find largest element from array without using conditional operator - GeeksforGeeks

Given an array of n-elements, we have to find the largest element among them without using any conditional operator like greater than or less than.

Examples:

```
Input : arr[] = {5, 7, 2, 9}
Output : Largest element = 9

Input : arr[] = {15, 0, 2, 15}
Output : Largest element = 15
```

First Approach (Use of Hashing) : To find the largest element from the array we may use the concept the of hashing where we should maintain a hash table of all element and then after processing all array element we should find the largest element in hash by simply traversing the hash table from end.

But there are some drawbacks of this approach like in case of very large elements maintaining a hash table is either not possible or not feasible.

Better Approach (Use of Bitwise AND) : Recently we have learn how to find the largest AND value pair from a given array. Also, we know that if we take bitwise AND of any number with INT_MAX (whose all bits are set bits) then the result will be that number itself. Now, using this property we will try to find the largest element from the array without any use conditional operator like greater than or less than.

For finding the largest element we will first insert an extra element i.e. INT_MAX in array, and after that we will try to find the maximum AND value of any pair from the array.

This obtained maximum value will contain AND value of INT_MAX and largest element of original array and is our required result.

Below is the implementation of above approach :

CPP

```
 // CPP Program to find largest element from array
#include <bits/stdc++.h>
using namespace std;

// Utility function to check number of elements
// having set msb as of pattern
int checkBit(int pattern, vector<int> arr, int n)
{
    int count = 0;
    for (int i = 0; i < n; i++)
        if ((pattern & arr[i]) == pattern)
            count++;
    return count;
}

// Function for finding maximum and value pair
int largest(int arr[], int n)
{
    // Create a vector of given array
    vector<int> v(arr, arr + n);

    // Insert INT_MAX and update n
    v.push_back(INT_MAX);
    n++;

    int res = 0;

    // Iterate over total of 30bits from msb to lsb
    for (int bit = 31; bit >= 0; bit--) {

        // Find the count of element having set msb
        int count = checkBit(res | (1 << bit), v, n);

        // if count | 1 != 1 set particular bit in result
        if (count | 1 != 1)
            res |= (1 << bit);
    }

    return res;
}
```

```
// Driver function
int main()
{
    int arr[] = { 4, 8, 6, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Largest element = " << largest(arr, n);
    return 0;
}
```

PHP

```
 <?php
// php Program to find largest
// element from array

// Utility function to check
// number of elements having
// set msb as of pattern
function checkBit($pattern,$arr,$n)
{
    $count = 0;
    for ($i = 0; $i < $n; $i++)
        if (($pattern & $arr[$i]) == $pattern)
            $count++;
    return $count;
}

// Function for finding
// maximum and value pair
function largest($arr, $n)
{
    $res = 0;

    // Iterate over total of
    // 30bits from msb to lsb
    for ($bit = 31; $bit >= 0; $bit--)
    {

        // Find the count of element
        // having set msb
        $count = checkBit($res | (1 << $bit),$arr, $n);

        // if count | 1 != 1 set
        // particular bit in result
        if ($count | 1 != 1)
            $res |= (1 << $bit);
    }
```

```
    return $res;
}

    // Driver code
    $arr = array( 4, 8, 6, 2 );
    $n = sizeof($arr) / sizeof($arr[0]);
    echo "Largest element = ". largest($arr, $n);

// This code is contributed by mits
?>
```

Output:

```
Largest element = 8
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/find-largest-element-array-without-using-conditional-operator/

Chapter 151

Find longest sequence of 1's in binary representation with one flip

Find longest sequence of 1's in binary representation with one flip - GeeksforGeeks

Give an integer n. We can flip exactly one bit. Write code to find the length of the longest sequence of 1 s you could create.

Examples:

```
Input : 1775
Output : 8
Binary representation of 1775 is 11011101111.
After flipping the highlighted bit, we get
consecutive 8 bits. 11011111111.

Input : 12
Output : 3

Input : 15
Output : 5

Input : 71
Output: 4

Binary representation of 71 is 1000111.
After flipping the highlighted bit, we get
consecutive 4 bits. 1001111.
```

A **simple solution** is to store binary representation of given number in a binary array. Once we have elements in binary array, we can apply methods discussed here.

An **efficient solution** is to walk through the bits in binary representation of given number. We keep track of current 1's sequence length and the previous 1's sequence length. When we see a zero, update previous Length:

1. If the next bit is a 1, previous Length should be set to current Length.
2. If the next bit is a 0, then we can't merge these sequences together. So, set previous Length to 0.

We update max length by comparing following two:

1. Current value of max-length
2. Current-Length + Previous-Length .

result = return max-length+1 (// add 1 for flip bit count)

.

Below is the implementation of above idea :

C++

```
 // C++ program to find maximum consecutive
// 1's in binary representation of a number
// after flipping one bit.
#include<bits/stdc++.h>
using namespace std;

int flipBit(unsigned a)
{
    /* If all bits are 1, binary representation
       of 'a' has all 1s */
    if (~a == 0)
        return 8*sizeof(int);

    int currLen = 0, prevLen = 0, maxLen = 0;
    while (a!= 0)
    {
        // If Current bit is a 1 then increament currLen++
        if ((a & 1) == 1)
            currLen++;

        // If Current bit is a 0 then check next bit of a
        else if ((a & 1) == 0)
        {
            /* Update prevLen to 0 (if next bit is 0)
            or currLen (if next bit is 1). */
```

```
            prevLen = (a & 2) == 0? 0 : currLen;

            // If two consecutively bits are 0
            // then currLen also will be 0.
            currLen = 0;
        }

        // Update maxLen if required
        maxLen = max(prevLen + currLen, maxLen);

        // Remove last bit (Right shift)
        a >>= 1;
    }

    // We can always have a sequence of
    // at least one 1, this is fliped bit
    return maxLen+1;
}

// Driver code
int main()
{
    // input 1
    cout << flipBit(13);
    cout << endl;

    // input 2
    cout << flipBit(1775);
    cout << endl;

    // input 3
    cout << flipBit(15);
    return 0;
}
```

Python3

```
 # Python3 program to find maximum
# consecutive 1's in binary
# representation of a number
# after flipping one bit.
def flipBit(a):

    # If all bits are 1,
    # binary representation
    # of 'a' has all 1s
    if (~a == 0):
        return 8 * sizeof();
```

```
    currLen = 0;
    prevLen = 0;
    maxLen = 0;
    while (a > 0):

        # If Current bit is a 1
        # then increament currLen++
        if ((a & 1) == 1):
            currLen += 1;

        # If Current bit is a 0
        # then check next bit of a
        elif ((a & 1) == 0):

            # Update prevLen to 0
            # (if next bit is 0)
            # or currLen (if next
            # bit is 1). */
            prevLen = 0 if((a & 2) == 0) else currLen;

            # If two consecutively bits
            # are 0 then currLen also
            # will be 0.
            currLen = 0;

        # Update maxLen if required
        maxLen = max(prevLen + currLen, maxLen);

        # Remove last bit (Right shift)
        a >>= 1;

    # We can always have a sequence
    # of at least one 1, this is
    # fliped bit
    return maxLen + 1;

# Driver code
# input 1
print(flipBit(13));

# input 2
print(flipBit(1775));

# input 3
print(flipBit(15));

# This code is contributed by mits
```

PHP

```
 <?php
// PHP program to find maximum consecutive
// 1's in binary representation of a number
// after flipping one bit.

function flipBit($a)
{
    /* If all bits are 1,
       binary representation
       of 'a' has all 1s */
    if (~$a == 0)
        return 8 * sizeof();

    $currLen = 0;
    $prevLen = 0;
    $maxLen = 0;
    while ($a!= 0)
    {

        // If Current bit is a 1
        // then increament currLen++
        if (($a & 1) == 1)
            $currLen++;

        // If Current bit is a 0
        // then check next bit of a
        else if (($a & 1) == 0)
        {

            /* Update prevLen to 0
               (if next bit is 0)
               or currLen (if next
               bit is 1). */
            $prevLen = ($a & 2) == 0? 0 : $currLen;

            // If two consecutively bits are 0
            // then currLen also will be 0.
            $currLen = 0;
        }

        // Update maxLen if required
        $maxLen = max($prevLen + $currLen, $maxLen);

        // Remove last bit (Right shift)
        $a >>= 1;
    }
```

```
    // We can always have a sequence of
    // at least one 1, this is fliped bit
    return $maxLen+1;
}

    // Driver code
    // input 1
    echo flipBit(13);
    echo "\n";

    // input 2
    echo flipBit(1775);
    echo "\n";

    // input 3
    echo flipBit(15);

// This code is contributed by aj_36
?>
```

Output :

```
4
8
5
```

Improved By : jit_t, Mithun Kumar

Source

https://www.geeksforgeeks.org/find-longest-sequence-1s-binary-representation-one-flip/

Chapter 152

Find missing number in another array which is shuffled copy

Find missing number in another array which is shuffled copy - GeeksforGeeks

Given an array 'arr1' of n positive integers. Contents of arr1[] are copied to another array 'arr2', but numbers are shuffled and one element is removed. Find the missing element(without using any extra space and in O(n) time complexity).

Examples :

```
Input : arr1[] = {4, 8, 1, 3, 7},
        arr2[] = {7, 4, 3, 1}
Output : 8

Input : arr1[] = {12, 10, 15, 23, 11, 30},
        arr2[] = {15, 12, 23, 11, 30}
Output : 10
```

A **simple solution** is to one by one consider every element of first array and search in second array. As soon as we find a missing element, we return. Time complexity of this solution is $O(n^2)$

An **efficient solution** is based on XOR. The combined occurrence of each element is twice, one in 'arr1' and other in 'arr2', except one element which only has a single occurrence in 'arr1'. We know that (a Xor a) = 0. So, simply XOR the elements of both the arrays. The result will be the missing number.

C++

```
 // C++ implementation to find the
// missing number in shuffled array
```

```
// C++ implementation to find the
// missing number in shuffled array
#include <bits/stdc++.h>
using namespace std;

// Returns the missing number
// Size of arr2[] is n-1
int missingNumber(int arr1[], int arr2[],
                                    int n)
{
    // Missing number 'mnum'
    int mnum = 0;

    // 1st array is of size 'n'
    for (int i = 0; i < n; i++)
        mnum = mnum ^ arr1[i];

    // 2nd array is of size 'n - 1'
    for (int i = 0; i < n - 1; i++)
        mnum = mnum ^ arr2[i];

    // Required missing number
    return mnum;
}

// Driver Code
int main()
{
    int arr1[] = {4, 8, 1, 3, 7};
    int arr2[] = {7, 4, 3, 1};
    int n = sizeof(arr1) / sizeof(arr1[0]);
    cout << "Missing number = "
        << missingNumber(arr1, arr2, n);
    return 0;
}
```

Java

```
 // Java implementation to find the
// missing number in shuffled array

class GFG
{
    // Returns the missing number
    // Size of arr2[] is n-1
    static int missingNumber(int arr1[],
                             int arr2[],
                                  int n)
```

```
    {
        // Missing number 'mnum'
        int mnum = 0;

        // 1st array is of size 'n'
        for (int i = 0; i < n; i++)
            mnum = mnum ^ arr1[i];

        // 2nd array is of size 'n - 1'
        for (int i = 0; i < n - 1; i++)
            mnum = mnum ^ arr2[i];

        // Required missing number
        return mnum;
    }

    // Driver Code
    public static void main (String[] args)
    {
        int arr1[] = {4, 8, 1, 3, 7};
        int arr2[] = {7, 4, 3, 1};
        int n = arr1.length;

        System.out.println("Missing number = "
            + missingNumber(arr1, arr2, n));
    }
}
```

Python3

```
 # Python3 implementation to find the
# missing number in shuffled array

# Returns the missing number
# Size of arr2[] is n - 1
def missingNumber(arr1, arr2, n):

    # missing number 'mnum'
    mnum = 0

    # 1st array is of size 'n'
    for i in range(n):
        mnum = mnum ^ arr1[i]

    # 2nd array is of size 'n - 1'
    for i in range(n - 1):
        mnum = mnum ^ arr2[i]
```

```
    # Required missing number
    return mnum

# Driver Code
arr1 = [4, 8, 1, 3, 7]
arr2= [7, 4, 3, 1]
n = len(arr1)
print("Missing number = ",
    missingNumber(arr1, arr2, n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# implementation to find the
// missing number in shuffled array
using System;

class GFG
{
    // Returns the missing number
    // Size of arr2[] is n-1
    static int missingNumber(int []arr1,
                             int []arr2,
                                  int n)
    {
        // Missing number 'mnum'
        int mnum = 0;

        // 1st array is of size 'n'
        for (int i = 0; i < n; i++)
            mnum = mnum ^ arr1[i];

        // 2nd array is of size 'n - 1'
        for (int i = 0; i < n - 1; i++)
            mnum = mnum ^ arr2[i];

        // Required missing number
        return mnum;
    }

    // Driver Code
    public static void Main ()
    {
        int []arr1 = {4, 8, 1, 3, 7};
        int []arr2 = {7, 4, 3, 1};
        int n = arr1.Length;
```

```
    Console.Write("Missing number = "
            + missingNumber(arr1, arr2, n));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP implementation to find the
// missing number in shuffled array
// PHP implementation to find the
// missing number in shuffled array

// Returns the missing number
// Size of arr2[] is n-1
function missingNumber($arr1, $arr2,
                                $n)
{

    // Missing number 'mnum'
    $mnum = 0;

    // 1st array is of size 'n'
    for ($i = 0; $i < $n; $i++)
        $mnum = $mnum ^ $arr1[$i];

    // 2nd array is of size 'n - 1'
    for ($i = 0; $i < $n - 1; $i++)
        $mnum = $mnum ^ $arr2[$i];

    // Required missing number
    return $mnum;
}

    // Driver Code
    $arr1 = array(4, 8, 1, 3, 7);
    $arr2 = array(7, 4, 3, 1);
    $n = count($arr1);
    echo "Missing number = "
        , missingNumber($arr1, $arr2, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
Missing number = 8
```

Complexity: O(n) time complexity and O(1) extra space.

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/find-missing-number-another-array-shuffled-copy/

Chapter 153

Find most significant set bit of a number

Find most significant set bit of a number - GeeksforGeeks

Given a number, find the most significant bit number which is set bit and which is in power of two

Examples:

```
Input : 10
Output : 8
Binary representation of 10 is 1010
The most significant bit corresponds
to decimal number 8.

Input : 18
Output : 16
```

A **simple solution** is to one by one divide n by 2 until it becomes 0 and increment a count while doing this. This count actually represents position of MSB.

C++

```
 // Simple CPP program to find MSB number
// for given n.
#include <iostream>
using namespace std;

int setBitNumber(int n)
{
    if (n == 0)
```

```
        return 0;

    int msb = 0;
    while (n != 0) {
        n = n / 2;
        msb++;
    }

    return (1 << msb);
}

// Driver code
int main()
{
    int n = 0;
    cout << setBitNumber(n);
    return 0;
}
```

Java

```
 // Simple Java rogram to find
// MSB number for given n.
import java.io.*;

class GFG
{
static int setBitNumber(int n)
{
    if (n == 0)
        return 0;

    int msb = 0;
    while (n != 0)
    {
        n = n / 2;
        msb++;
    }

    return (1 << msb);
}

// Driver code
public static void main (String[] args)
{
    int n = 0;
    System.out.println(setBitNumber(n));
}
```

```
}

// This code is contributed by ajit
```

Python3

```
 # Simple Python3 program
# to find MSB number
# for given n.
def setBitNumber(n):
    if (n == 0):
        return 0;

    msb = 0;
    while (n > 0):
        n = int(n / 2);
        msb += 1;

    return (1 << msb);

# Driver code
n = 0;
print(setBitNumber(n));

# This code is contributed
# by mits
```

C#

```
 // Simple C# rogram to find
// MSB number for given n.
using System;

class GFG
{
static int setBitNumber(int n)
{
    if (n == 0)
        return 0;

    int msb = 0;
    while (n != 0)
    {
        n = n / 2;
        msb++;
    }
```

```
    return (1 << msb);
}

// Driver code
static public void Main ()
{
    int n = 0;
    Console.WriteLine(setBitNumber(n));
}
}

// This code is contributed
// by akt_mit
```

PHP

```
 <?php
// Simple PhP program
// to find MSB number
// for given n.

function setBitNumber($n)
{
    if ($n == 0)
        return 0;

    $msb = 0;
    while ($n != 0)
    {
        $n = $n / 2;
        $msb++;
    }

    return (1 << $msb);
}

// Driver code
$n = 0;
echo setBitNumber($n);

// This code is contributed
// by akt_mit
?>
```

Output:

```
0
```

An **efficient solution** for a fixed size integer (say 32 bits) is to one by one set bits, then add 1 so that only the bit after MSB is set. Finally right shift by 1 and return answer. This solution does not require any condition checking.

C++

```
 // CPP program to find MSB number for given n.
#include <iostream>
using namespace std;

int setBitNumber(int n)
{
    // Below steps set bits after
    // MSB (including MSB)

    // Suppose n is 273 (binary
    // is 100010001). It does following
    // 100010001 | 010001000 = 110011001
    n |= n >> 1;

    // This makes sure 4 bits
    // (From MSB and including MSB)
    // are set. It does following
    // 110011001 | 001100110 = 111111111
    n |= n >> 2;

    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // Increment n by 1 so that
    // there is only one set bit
    // which is just before original
    // MSB. n now becomes 1000000000
    n = n + 1;

    // Return original MSB after shifting.
    // n now becomes 100000000
    return (n >> 1);
}

// Driver code
int main()
{
    int n = 273;
    cout << setBitNumber(n);
    return 0;
}
```

Java

```
 // Java program to find MSB
// number for given n.

class GFG {

    static int setBitNumber(int n)
    {

        // Below steps set bits after
        // MSB (including MSB)

        // Suppose n is 273 (binary
        // is 100010001). It does following
        // 100010001 | 010001000 = 110011001
        n |= n >> 1;

        // This makes sure 4 bits
        // (From MSB and including MSB)
        // are set. It does following
        // 110011001 | 001100110 = 111111111
        n |= n >> 2;

        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;

        // Increment n by 1 so that
        // there is only one set bit
        // which is just before original
        // MSB. n now becomes 1000000000
        n = n + 1;

        // Return original MSB after shifting.
        // n now becomes 100000000
        return (n >> 1);
    }

    // Driver code
    public static void main(String arg[])
    {
        int n = 273;
        System.out.print(setBitNumber(n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python program to find
# MSB number for given n.

def setBitNumber(n):

    # Below steps set bits after
    # MSB (including MSB)

    # Suppose n is 273 (binary
    # is 100010001). It does following
    # 100010001 | 010001000 = 110011001
    n |= n>>1

    # This makes sure 4 bits
    # (From MSB and including MSB)
    # are set. It does following
    # 110011001 | 001100110 = 111111111
    n |= n>>2

    n |= n>>4
    n |= n>>8
    n |= n>>16

    # Increment n by 1 so that
    # there is only one set bit
    # which is just before original
    # MSB. n now becomes 1000000000
    n = n + 1

    # Return original MSB after shifting.
    # n now becomes 100000000
    return (n >> 1)

# Driver code

n = 273
print(setBitNumber(n))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to find MSB number for given n.
using System;
```

```
class GFG {

    static int setBitNumber(int n)
    {

        // Below steps set bits after
        // MSB (including MSB)

        // Suppose n is 273 (binary
        // is 100010001). It does following
        // 100010001 | 010001000 = 110011001
        n |= n >> 1;

        // This makes sure 4 bits
        // (From MSB and including MSB)
        // are set. It does following
        // 110011001 | 001100110 = 111111111
        n |= n >> 2;

        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;

        // Increment n by 1 so that
        // there is only one set bit
        // which is just before original
        // MSB. n now becomes 1000000000
        n = n + 1;

        // Return original MSB after shifting.
        // n now becomes 100000000
        return (n >> 1);
    }

    // Driver code
    public static void Main()
    {
        int n = 273;
        Console.WriteLine(setBitNumber(n));
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
```

```
// PHP program to find
// MSB number for given n.

function setBitNumber($n)
{
    // Below steps set bits
    // after MSB (including MSB)

    // Suppose n is 273 (binary
    // is 100010001). It does
    // following 100010001 |
    // 010001000 = 110011001
    $n |= $n >> 1;

    // This makes sure 4 bits
    // (From MSB and including
    // MSB) are set. It does
    // following 110011001 |
    // 001100110 = 111111111
    $n |= $n >> 2;

    $n |= $n >> 4;
    $n |= $n >> 8;
    $n |= $n >> 16;

    // Increment n by 1 so
    // that there is only
    // one set bit which is
    // just before original
    // MSB. n now becomes
    // 1000000000
    $n = $n + 1;

    // Return original MSB
    // after shifting. n
    // now becomes 100000000
    return ($n >> 1);
}

// Driver code
$n = 273;
echo setBitNumber($n);

// This code is contributed
// by akt_mit
?>
```

Output:

256

Time complexity is O(1).

Another Approach: Given a number n. First, find the position of the most significant set bit and then compute the value of the number with set bit at k-th position.
Thanks Rohit Narayan for suggesting this method.

C++

```
 // CPP program to find MSB
// number for given n.
#include <bits/stdc++.h>
using namespace std;

int setBitNumber(int n)
{

    // To find the position
    // of the most significant
    // set bit
    int k = (int)(log2(n));

    // To return the the value
    // of the number with set
    // bit at k-th position
    return (int)(pow(2, k));
}

// Driver code
int main()
{
    int n = 273;
    cout << setBitNumber(n);
    return 0;
}
```

Java

```
 // Java program to find MSB
// number for given n.

class GFG {

    static int setBitNumber(int n)
    {
```

```
        // To find the position of the
        // most significant set bit
        int k = (int)(Math.log(n) / Math.log(2));

        // To return the the value of the number
        // with set bit at k-th position
        return (int)(Math.pow(2, k));
    }

    // Driver code
    public static void main(String arg[])
    {
        int n = 273;
        System.out.print(setBitNumber(n));
    }
}
```

Python3

```
 # Python program to find
# MSB number for given n.
import math

def setBitNumber(n):

    # To find the position of
    # the most significant
    # set bit
    k = int(math.log(n, 2))

    # To return the the value
    # of the number with set
    # bit at k-th position
    return 2**k

# Driver code
n = 273
print(setBitNumber(n))
```

C#

```
 // C# program to find MSB
// number for given n.
using System;

public class GFG {
```

```
    static int setBitNumber(int n)
    {

        // To find the position of the
        // most significant set bit
        int k = (int)(Math.Log(n) / Math.Log(2));

        // To return the the value of the number
        // with set bit at k-th position
        return (int)(Math.Pow(2, k));
    }

    // Driver code
    static public void Main()
    {
        int n = 273;
        Console.WriteLine(setBitNumber(n));
    }
}
```

PHP

```
 <?php
// PHP program to find MSB
// number for given n.

function setBitNumber($n)
{

    // To find the position
    // of the most significant
    // set bit
    $k =(int)(log($n,2));

    // To return the the value
    // of the number with set
    // bit at k-th position
    return (int)(pow(2, $k));
}

// Driver code
    $n = 273;
    echo setBitNumber($n);

// This code is contributed
// by jit_t.
?>
```

Output:

```
256
```

Improved By : Sam007, rohitnarayan, jit_t, Mithun Kumar

Source

https://www.geeksforgeeks.org/find-significant-set-bit-number/

Chapter 154

Find nth Magic Number

Find nth Magic Number - GeeksforGeeks

A magic number is defined as a number which can be expressed as a power of 5 or sum of unique powers of 5. First few magic numbers are 5, 25, 30(5 + 25), 125, 130(125 + 5), ….

Write a function to find the nth Magic number.

Example:

```
Input: n = 2
Output: 25

Input: n = 5
Output: 130
```

If we notice carefully the magic numbers can be represented as 001, 010, 011, 100, 101, 110 etc, where 001 is 0*pow(5,3) + 0*pow(5,2) + 1*pow(5,1). So basically we need to add powers of 5 for each bit set in given integer n.

Below is the implementation based on this idea.

C++

```
 // C++ program to find nth magic numebr
#include <bits/stdc++.h>
using namespace std;

// Function to find nth magic numebr
int nthMagicNo(int n)
{
    int pow = 1, answer = 0;
```

```
    // Go through every bit of n
    while (n)
    {
       pow = pow*5;

       // If last bit of n is set
       if (n & 1)
         answer += pow;

       // proceed to next bit
       n >>= 1;  // or n = n/2
    }
    return answer;
}

// Driver program to test above function
int main()
{
    int n = 5;
    cout << "nth magic number is " << nthMagicNo(n) << endl;
    return 0;
}
```

Java

```
 // Java program to find nth
// magic numebr
import java.io.*;

class GFG
{
  // Function to find nth magic number
  static int nthMagicNo(int n)
  {
     int pow = 1, answer = 0;

     // Go through every bit of n
     while (n != 0)
     {
       pow = pow*5;

       // If last bit of n is set
       if ((int)(n & 1) == 1)
         answer += pow;

       // proceed to next bit
       // or n = n/2
       n >>= 1;
```

```
    }
    return answer;
  }

  // Driver program to test
  // above function
  public static void main(String[] args)
  {
    int n = 5;
    System.out.println("nth magic" +
    " number is " + nthMagicNo(n));
  }
}


// This code is contributed by
// prerna saini
```

Python3

```
 # Python program to find nth magic numebr

# Function to find nth magic numebr
def nthMagicNo(n):

    pow = 1
    answer = 0

    # Go through every bit of n
    while (n):

        pow = pow*5

        # If last bit of n is set
        if (n & 1):
            answer += pow

        # proceed to next bit
        n >>= 1 # or n = n/2

    return answer


# Driver program to test above function
n = 5
print("nth magic number is", nthMagicNo(n))

# This code is contributed by
```

```
# Smitha Dinesh Semwal
```

C#

```
 // C# program to find nth
// magic numebr
using System;

public class GFG
{

// Function to find nth magic number
static int nthMagicNo(int n)
{
    int pow = 1, answer = 0;

    // Go through every bit of n
    while (n != 0)
    {
        pow = pow * 5;

        // If last bit of n is set
        if ((int)(n & 1) == 1)
            answer += pow;

        // proceed to next bit
        // or n = n/2
        n >>= 1;
    }
    return answer;
}

// Driver Code
public static void Main()
{
    int n = 5;
    Console.WriteLine("nth magic" +    " number is "
                      + nthMagicNo(n));

}
}

// This code is contributed by Sam007
```

PHP

```
 <?php
```

```
// PHP program to find nth
// magic number

// Function to find nth
// magic number
function nthMagicNo($n)
{
    $pow = 1;
    $answer = 0;

    // Go through every bit of n
    while ($n)
    {
    $pow = $pow * 5;

    // If last bit of n is set
    if ($n & 1)
        $answer += $pow;

    // proceed to next bit
    $n >>= 1; // or $n = $n/2
    }
    return $answer;
}

// Driver Code
$n = 5;
echo "nth magic number is ",
       nthMagicNo($n), "\n";

// This code is contributed by Ajit.
?>
```

Output :

```
 nth magic number is 130
```

Thanks to manrajsingh for suggesting above solution.

This article is contributed by **Abhay**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : Sam007, jit_t

Source

https://www.geeksforgeeks.org/find-nth-magic-number/

Chapter 155

Find number of pairs in an array such that their XOR is 0

Find number of pairs in an array such that their XOR is 0 - GeeksforGeeks

Given an array $A[]$ of size N. Find the number of pairs (i, j) such that A_i XOR A_j = 0, and 1 <= i < j <= N.

Examples :

```
Input : A[] = {1, 3, 4, 1, 4}
Output : 2
Explanation : Index (0, 3) and (2, 4)

Input : A[] = {2, 2, 2}
Output : 3
```

First Approach : **Sorting**

A_i XOR A_j = 0 is only satisfied when $A_i = A_j$. Therefore, we will first sort the array and then count the frequency of each element. By combinatorics, we can observe that if frequency of some element is $count$ then, it will contribute $\frac{count \cdot (count-1)}{2}$ to the answer.

Below is the implementation of above approach :

C++

```
// C++ program to find number
```

```
// of pairs in an array such
// that their XOR is 0
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the
// count
int calculate(int a[], int n)
{
    // Sorting the list using
    // built in function
    sort(a, a + n);

    int count = 1;
    int answer = 0;

    // Traversing through the
    // elements
    for (int i = 1; i < n; i++) {

        if (a[i] == a[i - 1]){

            // Counting frequency of each
            // elements
            count += 1;

        }
        else
        {
            // Adding the contribution of
            // the frequency to the answer
            answer = answer + (count * (count - 1)) / 2;
            count = 1;
        }
    }

    answer = answer + (count * (count - 1)) / 2;

    return answer;
}

// Driver Code
int main()
{

    int a[] = { 1, 2, 1, 2, 4 };
    int n = sizeof(a) / sizeof(a[0]);
```

```
    // Print the count
    cout << calculate(a, n);
    return 0;
}

// This article is contributed by Sahil_Bansall.
```

Java

```
 // Java program to find number
// of pairs in an array such
// that their XOR is 0
import java.util.*;

class GFG
{
    // Function to calculate
    // the count
    static int calculate(int a[], int n)
    {
        // Sorting the list using
        // built in function
        Arrays.sort(a);

        int count = 1;
        int answer = 0;

        // Traversing through the
        // elements
        for (int i = 1; i < n; i++)
        {

            if (a[i] == a[i - 1])
            {
                // Counting frequency of each
                // elements
                count += 1;

            }
            else
            {
                // Adding the contribution of
                // the frequency to the answer
                answer = answer + (count * (count - 1)) / 2;
                count = 1;
            }
        }
```

```
        answer = answer + (count * (count - 1)) / 2;

        return answer;
    }

    // Driver Code
    public static void main (String[] args)
    {
        int a[] = { 1, 2, 1, 2, 4 };
        int n = a.length;

        // Print the count
        System.out.println(calculate(a, n));
    }
}

// This code is contributed by Ansu Kumari.
```

Python3

```
 # Python3 program to find number of pairs
# in an array such that their XOR is 0

# Function to calculate the count
def calculate(a) :

    # Sorting the list using
    # built in function
    a.sort()

    count = 1
    answer = 0

    # Traversing through the elements
    for i in range(1, len(a)) :

        if a[i] == a[i - 1] :

            # Counting frequncy of each elements
            count += 1

        else :

            # Adding the contribution of
            # the frequency to the answer
            answer = answer + count * (count - 1) // 2
            count = 1
```

```
    answer = answer + count * (count - 1) // 2

    return answer


# Driver Code
if __name__ == '__main__':

    a = [1, 2, 1, 2, 4]

    # Print the count
    print(calculate(a))
```

C#

```
 // Java program to find number
// of pairs in an array such
// that their XOR is 0
using System;

class GFG
{
    // Function to calculate
    // the count
    static int calculate(int []a, int n)
    {
        // Sorting the list using
        // built in function
        Array.Sort(a);

        int count = 1;
        int answer = 0;

        // Traversing through the
        // elements
        for (int i = 1; i < n; i++)
        {

            if (a[i] == a[i - 1])
            {
                // Counting frequency of each
                // elements
                count += 1;

            }
            else
            {
                // Adding the contribution of
```

```
                // the frequency to the answer
                answer = answer + (count * (count - 1)) / 2;
                count = 1;
            }
        }

        answer = answer + (count * (count - 1)) / 2;

        return answer;
    }

    // Driver Code
    public static void Main ()
    {
        int []a = { 1, 2, 1, 2, 4 };
        int n = a.Length;

        // Print the count
        Console.WriteLine(calculate(a, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find number
// of pairs in an array such
// that their XOR is 0

// Function to calculate
// the count
function calculate($a, $n)
{

    // Sorting the list using
    // built in function
    sort($a);

    $count = 1;
    $answer = 0;

    // Traversing through the
    // elements
    for ($i = 1; $i < $n; $i++)
    {
```

```
        if ($a[$i] == $a[$i - 1])
        {

            // Counting frequency of
            // each elements
            $count += 1;

        }

        else
        {

            // Adding the contribution of
            // the frequency to the answer
            $answer = $answer + ($count *
                       ($count - 1)) / 2;
            $count = 1;
        }
    }

    $answer = $answer + ($count *
               ($count - 1)) / 2;

    return $answer;
}

    // Driver Code
    $a = array(1, 2, 1, 2, 4);
    $n = count($a);

    // Print the count
    echo calculate($a, $n);

// This code is contributed by anuj_67.
?>
```

Output :

```
2
```

Time Complexity : O(N Log N)

Second Approach : **Hashing (Index Mapping)**

Solution is handy, if we can count the frequency of each element in the array. Index mapping technique can be used to count the frequency of each element.

Below is the implementation of above approach :

C++

```
 // C++ program to find number of pairs
// in an array such that their XOR is 0
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the answer
int calculate(int a[]){

    // Finding the maximum of the array
    int *maximum = max_element(a, a + 5);

    // Creating frequency array
    // With initial value 0
    int frequency[*maximum + 1] = {0};

    // Traversing through the array
    for(int i = 0; i < (*maximum)+1; i++)
    {
        // Counting frequency
        frequency[a[i]] += 1;
    }
    int answer = 0;

    // Traversing through the frequency array
    for(int i = 0; i < (*maximum)+1; i++)
    {
        // Calculating answer
        answer = answer + frequency[i] * (frequency[i] - 1) ;
    }
    return answer/2;
}

// Driver Code
int main()
{
   int a[] = {1, 2, 1, 2, 4};

   // Function calling
   cout << (calculate(a));
}

// This code is contributed by Smitha
```

Python 3

```
 # Python3 program to find number of pairs
# in an array such that their XOR is 0

# Function to calculate the answer
def calculate(a) :

    # Finding the maximum of the array
    maximum = max(a)

    # Creating frequency array
    # With initial value 0
    frequency = [0 for x in range(maximum + 1)]

    # Traversing through the array
    for i in a :

        # Counting frequency
        frequency[i] += 1

    answer = 0

    # Traversing through the frequency array
    for i in frequency :

        # Calculating answer
        answer = answer + i * (i - 1) // 2

    return answer

# Driver Code
a = [1, 2, 1, 2, 4]
print(calculate(a))
```

PHP

Output :

```
2
```

Time Complexity : O(N)

Note : Index Mapping method can only be used when the numbers in the array are not large. In such cases, sorting method can be used.

Improved By : vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/find-number-pairs-array-xor-0/

Chapter 156

Find one extra character in a string

Find one extra character in a string - GeeksforGeeks

Given two strings which are of lengths n and n+1. The second string contains all the character of the first string, but there is one extra character. Your task to find the extra character in the second string.

Examples :

```
Input : string strA = "abcd";
        string strB = "cbdae";
Output : e
string B contain all the element
there is a one extra character which is e

Input : string strA = "kxml";
        string strB = "klxml";
Output : l
string B contain all the element
there is a one extra character which is l
```

Method 1(Brute Force):-
Check with two for loop.
Time Complexity:- O(n^2)
Space Complexity:- O(1).

Method 2(Hash Map):-
Create an empty hash table and insert all character of second string. Now remove all characters of first string. Remaining character is the extra character.

Time Complexity:- O(n)
Auxiliary Space:- O(n).

```
 // CPP program to find extra character in one
// string
#include <bits/stdc++.h>
using namespace std;

char findExtraCharcter(string strA, string strB)
{
    // store string values in map
    unordered_map<char, int> m1;

    // store second string in map with frequency
    for (int i = 0; i < strB.length(); i++)
        m1[strB[i]]--;

    // store first string in map with frequency
    for (int i = 0; i < strA.length(); i++)
        m1[strA[i]]--;

    for (auto h1 = m1.begin(); h1 != m1.end(); h1++) {

        // if the frequency is 1 then this
        // character is which is added extra
        if (h1->second == -1)
            return h1->first;
    }
}

int main()
{
    // given string
    string strA = "abcd";
    string strB = "cbdae";

    // find Extra Character
    cout << findExtraCharcter(strA, strB);
}
```

Output:

```
e
```

Method 3(Bits):-
traverse first and second string from starting with xor operation at the end you get the character which is extra.
Time Complexity:- O(n+n+1)
Space Complexity:- O(1).

C++

```
 // CPP program to find extra character in one
// string
#include <iostream>
using namespace std;

char findExtraCharcter(string strA, string strB)
{
    // result store the result
    int res = 0, i;

    // traverse string A till end and
    // xor with res
    for (i = 0; i < strA.length(); i++) {

        // xor with res
        res ^= strA[i];
    }

    // traverse string B till end and
    // xor with res
    for (i = 0; i < strB.length(); i++) {

        // xor with res
        res ^= strB[i];
    }

    // print result at the end
    return ((char)(res));
}

int main()
{
    // given string
    string strA = "abcd";
    string strB = "cbdad";
    cout << findExtraCharcter(strA, strB);
    return 0;
}
```

Java

```
 // Java program to find extra
// character in one string
import java.io.*;

class GFG {

    static char findExtraCharcter(String strA,
```

```
                                    String strB)
    {
        // result store the result
        int res = 0, i;

        // traverse string A till
        // end and xor with res
        for (i = 0; i < strA.length(); i++)
        {
            // xor with res
            res ^= strA.charAt(i);
        }

        // traverse string B till end and
        // xor with res
        for (i = 0; i < strB.length(); i++)
        {
            // xor with res
            res ^= strB.charAt(i);
        }

        // print result at the end
        return ((char)(res));
    }

    // Driver code
    public static void main(String args[])
    {
        // given string
        String strA = "abcd";
        String strB = "cbdad";
        System.out.println(findExtraCharcter(strA, strB));
    }
}

/*This code is contributed by Nikita Tiwari.*/
```

Python 3

```
 # Python 3 program to find
# extra character in one string

def findExtraCharcter(strA, strB) :

    # result store the result
    res = 0

    # traverse string A till
```

```
    # end and xor with res
    for i in range(0,len(strA)) :

        # xor with res
        res =res ^ (ord)(strA[i])

    # traverse string B till
    # end and xor with res
    for i in range(0,len(strB)) :

        # xor with res
        res = res ^ (ord)(strB[i])

    # print result at the end
    return ((chr)(res));

# given string
strA = "abcd"
strB = "cbdad"
print(findExtraCharcter(strA, strB))

# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# program to find extra character
// in one string
using System;

class GFG {

    static char findExtraCharcter(string strA,
                                  string strB)
    {
        // result store the result
        int res = 0, i;

        // traverse string A till end and
        // xor with res
        for (i = 0; i < strA.Length; i++) {

            // xor with res
            res ^= strA[i];
        }

        // traverse string B till end and
        // xor with res
        for (i = 0; i < strB.Length; i++) {
```

```
            // xor with res
            res ^= strB[i];
        }

        // print result at the end
        return ((char)(res));
    }

    // Driver Code
    public static void Main()
    {
        // given string
        string strA = "abcd";
        string strB = "cbdad";
        Console.WriteLine(
            findExtraCharcter(strA, strB));
    }
}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

PHP

```
 <?php
// PHP program to find extra character in
// one string

function findExtraCharcter($strA, $strB)
{
    // result store the result
    $res = 0;

    // traverse string A till end and
    // xor with res
    for ($i = 0; $i < strlen($strA); $i++)
    {

        // xor with res
        $res ^= ord($strA[$i]);
    }

    // traverse string B till end and
    // xor with res
    for ($i = 0; $i < strlen($strB); $i++)
    {
```

```
        // xor with res
        $res ^= ord($strB[$i]);
    }

    // print result at the end
    return $res;
}

// Driver code
$strA = "abcd";
$strB = "cbdad";
echo chr(findExtraCharcter($strA, $strB));

// This code is contributed by Manish Shaw
// (manishshaw1)
?>
```

Output:

```
d
```

Improved By : Nikita tiwari, manishshaw1, ashuk1992

Source

https://www.geeksforgeeks.org/find-one-extra-character-string/

Chapter 157

Find position of the only set bit

Find position of the only set bit - GeeksforGeeks

Given a number having only one '1' and all other '0's in its binary representation, find position of the only set bit. Source: Microsoft Interview | 18

The idea is to start from rightmost bit and one by one check value of every bit. Following is detailed algorithm.

1) If number is power of two then and then only its binary representation contains only one '1'. That's why check whether given number is power of 2 or not. If given number is not power of 2, then print error message and exit.

2) Initialize two variables; i = 1 (for looping) and pos = 1 (to find position of set bit)

3) Inside loop, do bitwise AND of i and number 'N'. If value of this operation is true, then "pos" bit is set, so break the loop and return position. Otherwise, increment "pos" by 1 and left shift i by 1 and repeat the procedure.

C

```
 // C program to find position of only set bit in a given number
#include <stdio.h>

// A utility function to check whether n is power of 2 or not. See http://goo.gl/17Arj
int isPowerOfTwo(unsigned n)
{  return n && (! (n & (n-1)) ); }

// Returns position of the only set bit in 'n'
int findPosition(unsigned n)
{
    if (!isPowerOfTwo(n))
        return -1;

    unsigned i = 1, pos = 1;
```

```
    // Iterate through bits of n till we find a set bit
    // i&n will be non-zero only when 'i' and 'n' have a set bit
    // at same position
    while (!(i & n))
    {
        // Unset current bit and set the next bit in 'i'
        i = i << 1;

        // increment position
        ++pos;
    }

    return pos;
}

// Driver program to test above function
int main(void)
{
    int n = 16;
    int pos = findPosition(n);
    (pos == -1)? printf("n = %d, Invalid numbern", n):
                 printf("n = %d, Position %d n", n, pos);

    n = 12;
    pos = findPosition(n);
    (pos == -1)? printf("n = %d, Invalid numbern", n):
                 printf("n = %d, Position %d n", n, pos);

    n = 128;
    pos = findPosition(n);
    (pos == -1)? printf("n = %d, Invalid numbern", n):
                 printf("n = %d, Position %d n", n, pos);

    return 0;
}
```

PHP

```
 <?php
// PHP program to find position of
// only set bit in a given number

// A utility function to check
// whether n is power of 2 or
// not. See http://goo.gl/17Arj
function isPowerOfTwo($n)
{
    return $n && (!($n & ($n - 1)));
```

```
}

// Returns position of the
// only set bit in 'n'
function findPosition($n)
{
    if (!isPowerOfTwo($n))
        return -1;

    $i = 1;
    $pos = 1;

    // Iterate through bits of n
    // till we find a set bit i&n
    // will be non-zero only when
    // 'i' and 'n' have a set bit
    // at same position
    while (!($i & $n))
    {
        // Unset current bit and
        // set the next bit in 'i'
        $i = $i << 1;

        // increment position
        ++$pos;
    }

    return $pos;
}

// Driver Code
$n = 16;
$pos = findPosition($n);
if (($pos == -1) == true)
        echo "n =", $n, "," ,
             " Invalid number" , "\n";
else
        echo "n = ", $n, ",",
             " Position ", $pos, "\n";

$n = 12;
$pos = findPosition($n);
if(($pos == -1) == true)
        echo "n = ", $n, ",",
             " Invalid number", "\n";
else
        echo "n =", $n, ",",
```

```
            " Position ", $pos, "\n";
$n = 128;
$pos = findPosition($n);
if(($pos == -1) == true)
        echo "n =", $n, ",",
             " Invalid number", "\n";
else
        echo "n = ", $n, ",",
             " Position ", $pos, "\n";

// This code is contributed by ajit
?>
```

Python3

```
 # Python3 program to find position of
# only set bit in a given number

# A utility function to check
# whether n is power of 2 or
# not.
def isPowerOfTwo(n):
    return (True if(n > 0 and
                    ((n & (n - 1)) > 0))
                  else False);

# Returns position of the
# only set bit in 'n'
def findPosition(n):
    if (isPowerOfTwo(n) == True):
        return -1;

    i = 1;
    pos = 1;

    # Iterate through bits of n
    # till we find a set bit i&n
    # will be non-zero only when
    # 'i' and 'n' have a set bit
    # at same position
    while ((i & n) == 0):

        # Unset current bit and
        # set the next bit in 'i'
        i = i << 1;

        # increment position
        pos += 1;
```

```
    return pos;

# Driver Code
n = 16;
pos = findPosition(n);
if (pos == -1):
    print("n =", n, ", Invalid number");
else:
    print("n =", n, ", Position ", pos);

n = 12;
pos = findPosition(n);
if (pos == -1):
    print("n =", n, ", Invalid number");
else:
    print("n =", n, ", Position ", pos);

n = 128;
pos = findPosition(n);
if (pos == -1):
    print("n =", n, ", Invalid number");
else:
    print("n =", n," , Position ", pos);

# This code is contributed by mits
```

Output :

```
n = 16, Position 5
n = 12, Invalid number
n = 128, Position 8
```

Following is **another method** for this problem. The idea is to one by one right shift the set bit of given number 'n' until 'n' becomes 0. Count how many times we shifted to make 'n' zero. The final count is position of the set bit.

C

```
 // C program to find position of only set bit in a given number
#include <stdio.h>

// A utility function to check whether n is power of 2 or not
int isPowerOfTwo(unsigned n)
{  return n && (! (n & (n-1)) ); }

// Returns position of the only set bit in 'n'
int findPosition(unsigned n)
```

```
{
    if (!isPowerOfTwo(n))
        return -1;

    unsigned count = 0;

    // One by one move the only set bit to right till it reaches end
    while (n)
    {
        n = n >> 1;

        // increment count of shifts
        ++count;
    }

    return count;
}

// Driver program to test above function
int main(void)
{
    int n = 0;
    int pos = findPosition(n);
    (pos == -1)? printf("n = %d, Invalid numbern", n):
                 printf("n = %d, Position %d n", n, pos);

    n = 12;
    pos = findPosition(n);
    (pos == -1)? printf("n = %d, Invalid numbern", n):
                 printf("n = %d, Position %d n", n, pos);

    n = 128;
    pos = findPosition(n);
    (pos == -1)? printf("n = %d, Invalid numbern", n):
                 printf("n = %d, Position %d n", n, pos);

    return 0;
}
```

PHP

```
 <?php
// PHP program to find position of
// only set bit in a given number

// A utility function to check
// whether n is power of 2 or not
function isPowerOfTwo($n)
```

```
{
    return $n && (! ($n & ($n - 1)));
}

// Returns position of the
// only set bit in 'n'
function findPosition($n)
{
    if (!isPowerOfTwo($n))
        return -1;

    $count = 0;

    // One by one move the only set
    // bit to right till it reaches end
    while ($n)
    {
        $n = $n >> 1;

        // increment count of shifts
        ++$count;
    }

    return $count;
}

// Driver Code
$n = 0;
$pos = findPosition($n);
if(($pos == -1) == true)
    echo "n = ", $n, ",",
         " Invalid number", "\n";
else
    echo "n = ", $n, ",",
         " Position ", $pos, "\n";

$n = 12;
$pos = findPosition($n);
if (($pos == -1) == true)
        echo "n = ", $n, ",",
             " Invalid number", "\n";
else
        echo "n = ", $n,
             " Position ", $pos, "\n";

$n = 128;
$pos = findPosition($n);
    if(($pos == -1) == true)
```

```
        echo "n = ", $n, ",",
             " Invalid number","\n";
else
        echo "n = ", $n, ",",
             " Position ", $pos, "\n";

// This code is contributed by ajit
?>
```

Output :

```
n = 0, Invalid number
n = 12, Invalid number
n = 128, Position 8
```

We can also use log base 2 to find the position. Thanks to Arunkumarfor suggesting this solution.

C

```
 #include <stdio.h>

unsigned int Log2n(unsigned int n)
{
   return (n > 1)? 1 + Log2n(n/2): 0;
}

int isPowerOfTwo(unsigned n)
{
    return n && (! (n & (n-1)) );
}

int findPosition(unsigned n)
{
    if (!isPowerOfTwo(n))
        return -1;
    return Log2n(n) + 1;
}

// Driver program to test above function
int main(void)
{
    int n = 0;
    int pos = findPosition(n);
    (pos == -1)? printf("n = %d, Invalid numbern", n):
                 printf("n = %d, Position %d n", n, pos);
```

```
    n = 12;
    pos = findPosition(n);
    (pos == -1)? printf("n = %d, Invalid numbern", n):
                 printf("n = %d, Position %d n", n, pos);

    n = 128;
    pos = findPosition(n);
    (pos == -1)? printf("n = %d, Invalid numbern", n):
                 printf("n = %d, Position %d n", n, pos);

    return 0;
}
```

Python3

```
 # Python program to find position
# of only set bit in a given number

def Log2n(n):
    if (n > 1):
        return (1 + Log2n(n/2))
    else:
        return 0

# A utility function to check
# whether n is power of 2 or not
def isPowerOfTwo(n):
    return n and (not (n & (n-1)) )

def findPosition(n):
    if (not isPowerOfTwo(n)):
        return -1
    return Log2n(n) + 1

# Driver program to test above function

n = 0
pos = findPosition(n)
if(pos == -1):
    print("n =", n, ", Invalid number")
else:
    print("n = ", n, ", Position ", pos)

n = 12
pos = findPosition(n)
if(pos == -1):
    print("n =", n, ", Invalid number")
else:
```

```
    print("n = ", n, ", Position ", pos)
n = 128
pos = findPosition(n)
if(pos == -1):
    print("n = ", n, ", Invalid number")
else:
    print("n = ", n, ", Position ", pos)

# This code is contributed
# by Sumit Sudhakar
```

PHP

```
 <?php
// PHP program to find position
// of only set bit in a given number
function Log2n($n)
{
return ($n > 1) ? 1 +
  Log2n($n / 2) : 0;
}

function isPowerOfTwo($n)
{
    return $n && (! ($n &
                     ($n - 1)));
}

function findPosition($n)
{
    if (!isPowerOfTwo($n))
        return -1;
    return Log2n($n) + 1;
}

// Driver Code
$n = 0;
$pos = findPosition($n);
if(($pos == -1) == true)
        echo "n =", $n, ", " ,
             " Invalid number", "\n";
else
        echo "n = ",$n, ",",
             " Position n", $pos,"\n";

$n = 12;
$pos = findPosition($n);
if(($pos == -1) == true)
```

```
            echo "n = ", $n,"," ,
                 " Invalid number","\n";
        else
            echo "n =", $n,",",
                 " Position", $pos, "\n";

// Driver Code
$n = 128;
$pos = findPosition($n);
if(($pos == -1) == true)
        echo "n = ", $n, ",",
             " Invalid number","\n";
else
        echo "n = ", $n, ",",
             " Position ",$pos, "\n";

// This code is contributed by aj_36
?>
```

Output :

```
n = 0, Invalid number
n = 12, Invalid number
n = 128, Position 8
```

This article is compiled by **Narendra Kangralkar**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : jit_t, Mithun Kumar

Source

https://www.geeksforgeeks.org/find-position-of-the-only-set-bit/

Chapter 158

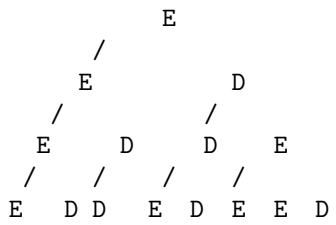
Find profession in a special family

Find profession in a special family - GeeksforGeeks

Consider a special family of Engineers and Doctors with following rules :

1. Everybody has two children.
2. First child of an Engineer is an Engineer and second child is a Doctor.
3. First child of an Doctor is Doctor and second child is an Engineer.
4. All generations of Doctors and Engineers start with Engineer.

We can represent the situation using below diagram:



Given level and position of a person in above ancestor tree, find the profession of the person.

Examples :

```
Input : level = 4, pos = 2
Output : Doctor

Input : level = 3, pos = 4
Output : Engineer
```

Method 1 (Recursive)
The idea is based on the fact that profession of a person depends on following two.

1. Profession of parent.
2. Position of node : If position of a node is odd, then its profession is same as its parent. Else profession is different from its parent.

We recursively find the profession of parent, then use point 2 above to find the profession of current node.

Below is implementation of above idea.

C++

```
 // C++ program to find profession of a person at
// given level and position.
#include<bits/stdc++.h>
using namespace std;

// Returns 'e' if profession of node at given level
// and position is engineer. Else doctor. The function
// assumes that given position and level have valid values.
char findProffesion(int level, int pos)
{
    // Base case
    if (level == 1)
        return 'e';

    // Recursively find parent's profession. If parent
    // is a doctar, this node will be a doctal if it is
    // at odd position and an engineer if at even position
    if (findProffesion(level-1, (pos+1)/2) == 'd')
        return (pos%2)? 'd' : 'e';

    // If parent is an engineer, then current node will be
    // an enginner if at add position and doctor if even
    // position.
    return (pos%2)?  'e' : 'd';
}

// Driver code
int main(void)
{
    int level = 4, pos = 2;
    (findProffesion(level, pos) == 'e')? cout << "Engineer"
                                       : cout << "Doctor" ;
    return 0;
}
```

Java

```
 // Java program to find
// profession of a person
// at given level and position
import java.io.*;

class GFG
{

// Returns 'e' if profession
// of node at given level
// and position is engineer.
// Else doctor. The function
// assumes that given position
// and level have valid values.
static char findProffesion(int level,
                           int pos)
{
    // Base case
    if (level == 1)
        return 'e';

    // Recursively find parent's
    // profession. If parent
    // is a doctar, this node
    // will be a doctal if it
    // is at odd position and an
    // engineer if at even position
    if (findProffesion(level - 1,
                     (pos + 1) / 2) == 'd')
        return (pos % 2 > 0) ?
                         'd' : 'e';

    // If parent is an engineer,
    // then current node will be
    // an enginner if at add
    // position and doctor if even
    // position.
    return (pos % 2 > 0) ?
                     'e' : 'd';
}

// Driver code
public static void main (String[] args)
{
    int level = 4, pos = 2;
    if(findProffesion(level,
```

```
                    pos) == 'e')
    System.out.println("Engineer");
    else
    System.out.println("Doctor");
}
}

// This code is contributed
// by anuj_67.
```

C#

```
 // C# program to find
// profession of a person
// at given level and position
using System;

class GFG
{

// Returns 'e' if profession
// of node at given level
// and position is engineer.
// Else doctor. The function
// assumes that given position
// and level have valid values.
static char findProffesion(int level,
                           int pos)
{
    // Base case
    if (level == 1)
        return 'e';

    // Recursively find parent's
    // profession. If parent
    // is a doctar, this node
    // will be a doctal if it
    // is at odd position and an
    // engineer if at even position
    if (findProffesion(level - 1,
                    (pos + 1) / 2) == 'd')
        return (pos % 2 > 0) ?
                        'd' : 'e';

    // If parent is an engineer,
    // then current node will be
    // an enginner if at add
    // position and doctor if even
```

```
    // position.
    return (pos % 2 > 0) ?
                    'e' : 'd';
}

// Driver code
public static void Main ()
{
    int level = 4, pos = 2;
    if(findProffesion(level,
                   pos) == 'e')
    Console.WriteLine("Engineer");
    else
    Console.WriteLine("Doctor");
}
}

// This code is contributed
// by anuj_67.
```

PHP

```
 <?php
// PHP program to find profession
// of a person at given level
// and position.

// Returns 'e' if profession of
// node at given level and position
// is engineer. Else doctor. The
// function assumes that given
// position and level have valid values.
function findProffesion($level, $pos)
{
    // Base case
    if ($level == 1)
        return 'e';

    // Recursively find parent's
    // profession. If parent is
    // a Doctor, this node will
    // be a doctor if it is at
    // odd position and an engineer
    // if at even position
    if (findProffesion($level - 1,
                     ($pos + 1) / 2) == 'd')
        return ($pos % 2) ? 'd' : 'e';
```

```
    // If parent is an engineer, then
    // current node will be an enginner
    // if at odd position and doctor
    // if even position.
    return ($pos % 2) ? 'e' : 'd';
}

// Driver code
$level = 4; $pos = 2;
if((findProffesion($level,
                   $pos) == 'e') == true)
    echo "Engineer";
else
    echo "Doctor" ;

// This code is contributed by ajit
?>
```

Output :

```
Doctor
```

Method 2 (Using Bitwise Operators)

```
Level 1: E
Level 2: ED
Level 3: EDDE
Level 4: EDDEDEED
Level 5: EDDEDEEDDEEDEDDE
```

Level input isn't necessary (if we ignore max position limit) because first elements are same.

The result is based on count of 1's in binary representation of position minus one. If count of 1's is even then result is Engineer, else then Doctor.

And of course position limit is 2^(Level-1)

C++

```
 // C++ program to find profession of a person at
// given level and position.
#include<bits/stdc++.h>
using namespace std;
```

```
/* Function to get no of set bits in binary
   representation of passed binary no. */
int countSetBits(int n)
{
    int count = 0;
    while (n)
    {
      n &= (n-1) ;
      count++;
    }
    return count;
}

// Returns 'e' if profession of node at given level
// and position is engineer. Else doctor. The function
// assumes that given position and level have valid values.
char findProffesion(int level, int pos)
{
    // Count set bits in 'pos-1'
    int c = countSetBits(pos-1);

    // If set bit count is odd, then doctor, else engineer
    return (c%2)?  'd' : 'e';
}

// Driver code
int main(void)
{
    int level = 3, pos = 4;
    (findProffesion(level, pos) == 'e')? cout << "Engineer"
                                       : cout << "Doctor" ;
    return 0;
}
```

PHP

```
 <?php
// PHP program to find profession
// of a person at given level and position.

// 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;
}

// Returns 'e' if profession of
// node at given level and position
// is engineer. Else doctor. The
// function assumes that given
// position and level have valid values.
function findProffesion($level, $pos)
{
    // Count set bits in 'pos-1'
    $c = countSetBits($pos - 1);

    // If set bit count is odd,
    // then doctor, else engineer
    return ($c % 2) ? 'd' : 'e';
}

// Driver Code
$level = 3;
$pos = 4;
if((findProffesion($level,
                   $pos) == 'e') == true)
        echo "Engineer \n";
    else
        echo "Doctor \n";

// This code is contributed by aj_36
?>
```

Output :

```
Engineer
```

Thanks to Furkan Uslu for suggesting this method.

Improved By : jit_t, vt_m

Source

https://www.geeksforgeeks.org/find-profession-in-a-hypothetical-special-situation/

Chapter 159

Find smallest number n such that n XOR n+1 equals to given k.

Find smallest number n such that n XOR n+1 equals to given k. - GeeksforGeeks

You are given a positive number k, we need to find a positive integer n, such that XOR of n and n+1 is equal to k. If no such n exist then print -1.

Examples:

```
Input : 3
Output : 1

Input : 7
Output : 3

Input : 6
Output : -1
```

Below are two cases when we do n XOR (n+1) for a number n.

Case 1 : n is even. Last bit of n is 0 and last bit of (n+1) is 1. Rest of the bits are same in both. So XOR would always be 1 if n is even.

Case : n is odd Last bit in n is 1. And in n+1, last bit is 0. But in this case there may be more bits which differ due to carry. The carry continues to propagate to left till we find first 0 bit. So n XOR n+1 will we 2^i-1 where i is the position of first 0 bit in n from left. So, we can say that if k is of form 2^i-1 then we will have our answer as k/2.

Finally our steps are:

```
If we have k=1, answer = 2 [We need smallest positive n]
Else If k is of form 2^i-1, answer = k/2,
else, answer = -1
```

C++

```
 // CPP to find n such that XOR of n and n+1
// is equals to given n
#include <bits/stdc++.h>
using namespace std;

// function to return the required n
int xorCalc(int k)
{
    if (k == 1)
        return 2;

    // if k is of form 2^i-1
    if (((k + 1) & k) == 0)
        return k / 2;

    return 1;
}

// driver program
int main()
{
    int k = 31;
    cout << xorCalc(k);
    return 0;
}
```

Java

```
 // Java to find n such that XOR of n and n+1
// is equals to given n
class GFG
{

    // function to return the required n
    static int xorCalc(int k)
    {
        if (k == 1)
            return 2;

        // if k is of form 2^i-1
```

```
        if (((k + 1) & k) == 0)
            return k / 2;

        return 1;
    }

    // Driver code
    public static void main (String[] args)
    {
        int k = 31;

        System.out.println(xorCalc(k));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # python to find n such that
# XOR of n and n+1 is equals
# to given n

# function to return the
# required n
def xorCalc(k):
    if (k == 1):
        return 2

    # if k is of form 2^i-1
    if (((k + 1) & k) == 0):
        return k / 2

    return 1;


# driver program
k = 31
print(int(xorCalc(k)))

# This code is contributed
# by Sam007
```

C#

```
 // C# to find n such that XOR
// of n and n+1 is equals to
```

```
// given n
using System;

class GFG
{

    // function to return the required
    // n
    static int xorCalc(int k)
    {
        if (k == 1)
            return 2;

        // if k is of form 2^i-1
        if (((k + 1) & k) == 0)
            return k / 2;

        return 1;
    }

    // Driver code
    public static void Main ()
    {
        int k = 31;

        Console.WriteLine(xorCalc(k));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP to find n such
// that XOR of n and n+1
// is equals to given n

// function to return
// the required n
function xorCalc($k)
{
    if ($k == 1)
        return 2;

    // if k is of form 2^i-1
    if ((($k + 1) & $k) == 0)
        return floor($k / 2);
```

```
    return 1;
}

// Driver Code
$k = 31;
echo xorCalc($k);

// This code is contributed by vt_m.
?>
```

Output:

```
15
```

Improved By : vt_m, Sam007

Source

https://www.geeksforgeeks.org/find-smallest-number-n-n-xor-n1-equals-given-k/

Chapter 160

Find the Number Occurring Odd Number of Times

Find the Number Occurring Odd Number of Times - GeeksforGeeks

Given an array of positive integers. All numbers occur even number of times except one number which occurs odd number of times. Find the number in O(n) time & constant space.

Examples :

```
Input : arr = {1, 2, 3, 2, 3, 1, 3}
Output : 3

Input : arr = {5, 7, 2, 7, 5, 2, 5}
Output : 5
```

A **Simple Solution** is to run two nested loops. The outer loop picks all elements one by one and inner loop counts number of occurrences of the element picked by outer loop. Time complexity of this solution is $O(n^2)$.

Below is the implementation of the brute force approach :

C++

```
 // C++ program to find the element
// occurring odd number of times
#include<bits/stdc++.h>
using namespace std;

// Funtion to find the element
// occurring odd number of times
int getOddOccurrence(int arr[], int arr_size)
```

```
{
    for (int i = 0; i < arr_size; i++) {

        int count = 0;

        for (int j = 0; j < arr_size; j++)
        {
            if (arr[i] == arr[j])
                count++;
        }
        if (count % 2 != 0)
            return arr[i];
    }
    return -1;
}

// driver code
int main()
    {
        int arr[] = { 2, 3, 5, 4, 5, 2,
                      4, 3, 5, 2, 4, 4, 2 };
        int n = sizeof(arr) / sizeof(arr[0]);

        // Function calling
        cout << getOddOccurrence(arr, n);

        return 0;
    }
```

Java

```
 // Java program to find the element occurring
// odd number of times
class OddOccurrence {

    // funtion to find the element occurring odd
    // number of times
    static int getOddOccurrence(int arr[], int arr_size)
    {
        int i;
        for (i = 0; i < arr_size; i++) {
            int count = 0;
            for (int j = 0; j < arr_size; j++) {
                if (arr[i] == arr[j])
                    count++;
            }
            if (count % 2 != 0)
                return arr[i];
```

```
        }
        return -1;
    }

    // driver code
    public static void main(String[] args)
    {
        int arr[] = new int[]{ 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 };
        int n = arr.length;
        System.out.println(getOddOccurrence(arr, n));
    }
}
// This code has been contributed by Kamal Rawal
```

Python3

```
 # Python program to find the element occurring
# odd number of times

# funtion to find the element occurring odd
# number of times
def getOddOccurrence(arr, arr_size):

    for i in range(0,arr_size):
        count = 0
        for j in range(0, arr_size):
            if arr[i] == arr[j]:
                count+=1

        if (count % 2 != 0):
            return arr[i]

    return -1


# driver code
arr = [2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 ]
n = len(arr)
print(getOddOccurrence(arr, n))

# This code has been contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to find the element
// occurring odd number of times
```

```
using System;

class GFG
{
    // Funtion to find the element
    // occurring odd number of times
    static int getOddOccurrence(int []arr, int arr_size)
    {
        for (int i = 0; i < arr_size; i++) {
            int count = 0;

            for (int j = 0; j < arr_size; j++) {
                if (arr[i] == arr[j])
                    count++;
            }
            if (count % 2 != 0)
                return arr[i];
        }
        return -1;
    }

    // Driver code
    public static void Main()
    {
        int []arr = { 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 };
        int n = arr.Length;
        Console.Write(getOddOccurrence(arr, n));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to find the
// element occurring odd
// number of times

// Function to find the element
// occurring odd number of times
function getOddOccurrence(&$arr, $arr_size)
{
    $count = 0;
    for ($i = 0;
         $i < $arr_size; $i++)
    {
```

```
        for ($j = 0;
             $j < $arr_size; $j++)
        {
            if ($arr[$i] == $arr[$j])
                $count++;
        }
        if ($count % 2 != 0)
            return $arr[$i];
    }
    return -1;
}

// Driver code
$arr = array(2, 3, 5, 4, 5, 2,
             4, 3, 5, 2, 4, 4, 2);
$n = sizeof($arr);

// Function calling
echo(getOddOccurrence($arr, $n));

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output :

```
5
```

A **Better Solution** is to use Hashing. Use array elements as key and their counts as value. Create an empty hash table. One by one traverse the given array elements and store counts. Time complexity of this solution is O(n). But it requires extra space for hashing.

Program :

Java

```
 //Java program to find the element occurring odd
// number of times
import java.io.*;
import java.util.HashMap;

class OddOccurrence
{
    // funtion to find the element occurring odd
    // number of times
    static int getOddOccurrence(int arr[], int n)
```

```
    {
        HashMap<Integer,Integer> hmap = new HashMap<>();

        // Putting all elements into the HashMap
        for(int i = 0; i < n; i++)
        {
            if(hmap.containsKey(arr[i]))
            {
                int val = hmap.get(arr[i]);

                // If array element is already present then
                // increase the count of that element.
                hmap.put(arr[i], val + 1);
            }
            else

                // if array element is not present then put
                // element into the HashMap and initialize
                // the count to one.
                hmap.put(arr[i], 1);
        }

        // Checking for odd occurrence of each element present
        // in the HashMap
        for(Integer a:hmap.keySet())
        {
            if(hmap.get(a) % 2 != 0)
                return a;
        }
        return -1;
    }

    // driver code
    public static void main(String[] args)
    {
        int arr[] = new int[]{2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};
        int n = arr.length;
        System.out.println(getOddOccurrence(arr, n));
    }
}
// This code is contributed by Kamal Rawal
```

Output :

```
5
```

The **Best Solution** is to do bitwise XOR of all the elements. XOR of all elements gives

us odd occurring element. Please note that XOR of two elements is 0 if both elements are same and XOR of a number x with 0 is x.

Below is the implementation of the above approach.

C++

```
 // C++ program to find the element
// occurring odd number of times
#include <bits/stdc++.h>
using namespace std;

// Function to find element occurring
// odd number of times
int getOddOccurrence(int ar[], int ar_size)
{
    int res = 0;
    for (int i = 0; i < ar_size; i++)
        res = res ^ ar[i];

    return res;
}

/* Diver function to test above function */
int main()
{
    int ar[] = {2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};
    int n = sizeof(ar)/sizeof(ar[0]);

    // Function calling
    cout << getOddOccurrence(ar, n);

    return 0;
}
```

C

```
// C program to find the element
// occurring odd number of times
#include <stdio.h>

// Function to find element occurring
// odd number of times
int getOddOccurrence(int ar[], int ar_size)
{
    int res = 0;
```

```
    for (int i = 0; i < ar_size; i++)
        res = res ^ ar[i];

    return res;
}

/* Diver function to test above function */
int main()
{
    int ar[] = {2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};
    int n = sizeof(ar) / sizeof(ar[0]);

    // Function calling
    printf("%d", getOddOccurrence(ar, n));
    return 0;
}
```

Java

```
 //Java program to find the element occurring odd number of times

class OddOccurance
{
    int getOddOccurrence(int ar[], int ar_size)
    {
        int i;
        int res = 0;
        for (i = 0; i < ar_size; i++)
        {
            res = res ^ ar[i];
        }
        return res;
    }

    public static void main(String[] args)
    {
        OddOccurance occur = new OddOccurance();
        int ar[] = new int[]{2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};
        int n = ar.length;
        System.out.println(occur.getOddOccurrence(ar, n));
    }
}
// This code has been contributed by Mayank Jaiswal
```

Python

```
# Python program to find the element occurring odd number of times

def getOddOccurrence(arr):

    # Initialize result
    res = 0

    # Traverse the array
    for element in arr:
        # XOR with the result
        res = res ^ element

    return res

# Test array
arr = [ 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2]

print "%d" % getOddOccurrence(arr)
```

C#

```
 // C# program to find the element
// occurring odd number of times
using System;

class GFG
{
    // Funtion to find the element
    // occurring odd number of times
    static int getOddOccurrence(int []arr, int arr_size)
    {
        int res = 0;
        for (int i = 0; i < arr_size; i++)
        {
            res = res ^ arr[i];
        }
        return res;
    }

    // Driver code
    public static void Main()
    {
        int []arr = { 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 };
        int n = arr.Length;
        Console.Write(getOddOccurrence(arr, n));
    }
}
```

```
// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to find the
// element occurring odd
// number of times

// Function to find element
// occurring odd number of times
function getOddOccurrence(&$ar, $ar_size)
{
    $res = 0;
    for ($i = 0; $i < $ar_size; $i++)
        $res = $res ^ $ar[$i];

    return $res;
}

// Driver Code
$ar = array(2, 3, 5, 4, 5, 2,
            4, 3, 5, 2, 4, 4, 2);
$n = sizeof($ar);

// Function calling
echo(getOddOccurrence($ar, $n));

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output :

```
5
```

Time Complexity: O(n)

Improved By : Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/find-the-number-occurring-odd-number-of-times/

Chapter 161

Find the element that appears once

Find the element that appears once - GeeksforGeeks

Given an array where every element occurs three times, except one element which occurs only once. Find the element that occurs once. Expected time complexity is O(n) and O(1) extra space.
Examples :

```
Input: arr[] = {12, 1, 12, 3, 12, 1, 1, 2, 3, 3}
Output: 2
```

We can use sorting to do it in O(nLogn) time. We can also use hashing, it has the worst case time complexity of O(n), but requires extra space.

The idea is to use bitwise operators for a solution that is O(n) time and uses O(1) extra space. The solution is not easy like other XOR based solutions, because all elements appear odd number of times here. The idea is taken from here.

Run a loop for all elements in array. At the end of every iteration, maintain following two values.

ones: The bits that have appeared 1st time or 4th time or 7th time .. etc.

twos: The bits that have appeared 2nd time or 5th time or 8th time .. etc.

Finally, we return the value of 'ones'

How to maintain the values of 'ones' and 'twos'?
'ones' and 'twos' are initialized as 0. For every new element in array, find out the common set bits in the new element and previous value of 'ones'. These common set bits are actually the bits that should be added to 'twos'. So do bitwise OR of the common set bits with 'twos'. 'twos' also gets some extra bits that appear third time. These extra bits are removed later.

Update ‘ones’ by doing XOR of new element with previous value of ‘ones’. There may be some bits which appear 3rd time. These extra bits are also removed later.

Both ‘ones’ and ‘twos’ contain those extra bits which appear 3rd time. Remove these extra bits by finding out common set bits in ‘ones’ and ‘twos’.

C/C++

```
 #include <stdio.h>

int getSingle(int arr[], int n)
{
    int ones = 0, twos = 0 ;

    int common_bit_mask;

    // Let us take the example of {3, 3, 2, 3} to understand this
    for( int i=0; i< n; i++ )
    {
        /* The expression "one & arr[i]" gives the bits that are
           there in both 'ones' and new element from arr[].  We
           add these bits to 'twos' using bitwise OR

           Value of 'twos' will be set as 0, 3, 3 and 1 after 1st,
           2nd, 3rd and 4th iterations respectively */
        twos  = twos | (ones & arr[i]);


        /* XOR the new bits with previous 'ones' to get all bits
           appearing odd number of times

           Value of 'ones' will be set as 3, 0, 2 and 3 after 1st,
           2nd, 3rd and 4th iterations respectively */
        ones  = ones ^ arr[i];


        /* The common bits are those bits which appear third time
           So these bits should not be there in both 'ones' and 'twos'.
           common_bit_mask contains all these bits as 0, so that the bits can
           be removed from 'ones' and 'twos'

           Value of 'common_bit_mask' will be set as 00, 00, 01 and 10
           after 1st, 2nd, 3rd and 4th iterations respectively */
        common_bit_mask = ~(ones & twos);


        /* Remove common bits (the bits that appear third time) from 'ones'

           Value of 'ones' will be set as 3, 0, 0 and 2 after 1st,
```

```
           2nd, 3rd and 4th iterations respectively */
        ones &= common_bit_mask;


        /* Remove common bits (the bits that appear third time) from 'twos'

           Value of 'twos' will be set as 0, 3, 1 and 0 after 1st,
           2nd, 3rd and 4th itearations respectively */
        twos &= common_bit_mask;

        // uncomment this code to see intermediate values
        //printf (" %d %d n", ones, twos);
    }

    return ones;
}

int main()
{
    int arr[] = {3, 3, 2, 3};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("The element with single occurrence is %d ",
            getSingle(arr, n));
    return 0;
}
```

Java

```
 //JAVA code to find the element
//that occur only once

class GFG
{
    // Method to find the element that occur only once
    static int getSingle(int arr[], int n)
    {
        int ones = 0, twos = 0;
        int common_bit_mask;

        for(int i=0; i<n; i++ )
        {
            /*"one & arr[i]" gives the bits that are there in
            both 'ones' and new element from arr[]. We
            add these bits to 'twos' using bitwise OR*/
            twos = twos | (ones & arr[i]);

            /*"one & arr[i]" gives the bits that are
            there in both 'ones' and new element from arr[].
```

```
            We add these bits to 'twos' using bitwise OR*/
            ones = ones ^ arr[i];

            /* The common bits are those bits which appear third time
            So these bits should not be there in both 'ones' and 'twos'.
            common_bit_mask contains all these bits as 0, so that the bits can
            be removed from 'ones' and 'twos'*/
            common_bit_mask = ~(ones & twos);

            /*Remove common bits (the bits that appear third time) from 'ones'*/
            ones &= common_bit_mask;

            /*Remove common bits (the bits that appear third time) from 'twos'*/
            twos &= common_bit_mask;
        }
        return ones;
    }

    //  Driver method
    public static void main(String args[])
    {
        int arr[] = {3, 3, 2, 3};
        int n = arr.length;
        System.out.println("The element with single occurrence is " + getSingle(arr, n));
    }
}
// Code contributed by Rishab Jain
```

Python3

```
 # Python3 code to find the element that
# appears once

def getSingle(arr, n):
    ones = 0
    twos = 0

    for i in range(n):
        # one & arr[i]" gives the bits that
        # are there in both 'ones' and new
        # element from arr[]. We add these
        # bits to 'twos' using bitwise OR
        twos = twos | (ones & arr[i])

        # one & arr[i]" gives the bits that
        # are there in both 'ones' and new
        # element from arr[]. We add these
        # bits to 'twos' using bitwise OR
```

```
        ones = ones ^ arr[i]

        # The common bits are those bits
        # which appear third time. So these
        # bits should not be there in both
        # 'ones' and 'twos'. common_bit_mask
        # contains all these bits as 0, so
        # that the bits can be removed from
        # 'ones' and 'twos'
        common_bit_mask = ~(ones & twos)

        # Remove common bits (the bits that
        # appear third time) from 'ones'
        ones &= common_bit_mask

        # Remove common bits (the bits that
        # appear third time) from 'twos'
        twos &= common_bit_mask
    return ones

# driver code
arr = [3, 3, 2, 3]
n = len(arr)
print("The element with single occurrence is ",
        getSingle(arr, n))

# This code is contributed by "Abhishek Sharma 44"
```

C#

```
 // C# code to find the element
// that occur only once
using System;
class GFG
{
    // Method to find the element
    // that occur only once
    static int getSingle(int []arr, int n)
    {
        int ones = 0, twos = 0;
        int common_bit_mask;

        for(int i = 0; i < n; i++ )
        {
            // "one & arr[i]" gives the bits
            // that are there in both 'ones'
            // and new element from arr[].
            // We add these bits to 'twos'
```

```
            // using bitwise OR
            twos = twos | (ones & arr[i]);

            // "one & arr[i]" gives the bits
            // that are there in both 'ones'
            // and new element from arr[].
            // We add these bits to 'twos'
            // using bitwise OR
            ones = ones ^ arr[i];

            // The common bits are those bits
            // which appear third time So these
            // bits should not be there in both
            // 'ones' and 'twos'. common_bit_mask
            // contains all these bits as 0,
            // so that the bits can be removed
            // from 'ones' and 'twos'
            common_bit_mask = ~(ones & twos);

            // Remove common bits (the bits that
            // appear third time) from 'ones'
            ones &= common_bit_mask;

            // Remove common bits (the bits that
            // appear third time) from 'twos'
            twos &= common_bit_mask;
        }
        return ones;
    }

    // Driver code
    public static void Main()
    {
        int []arr = {3, 3, 2, 3};
        int n = arr.Length;
        Console.WriteLine("The element with single" +
            "occurrence is " + getSingle(arr, n));
    }
}

// This Code is contributed by vt_m.
```

PHP

```
 <?php

function getSingle($arr, $n)
{
```

```
$ones = 0; $twos = 0 ;

$common_bit_mask;

// Let us take the example of
// {3, 3, 2, 3} to understand this
for($i = 0; $i < $n; $i++ )
{
    /* The expression "one & arr[i]"
    gives the bits that are there in
    both 'ones' and new element from
    arr[]. We add these bits to 'twos'
    using bitwise OR
    Value of 'twos' will be set as 0,
    3, 3 and 1 after 1st, 2nd, 3rd
    and 4th iterations respectively */
    $twos = $twos | ($ones & $arr[$i]);


    /* XOR the new bits with previous
    'ones' to get all bits appearing
    odd number of times

    Value of 'ones' will be set as 3,
    0, 2 and 3 after 1st, 2nd, 3rd and
    4th iterations respectively */
    $ones = $ones ^ $arr[$i];

    /* The common bits are those bits
    which appear third time. So these
    bits should not be there in both
    'ones' and 'twos'. common_bit_mask
    contains all these bits as 0, so
    that the bits can be removed from
    'ones' and 'twos'

    Value of 'common_bit_mask' will be
    set as 00, 00, 01 and 10 after 1st,
    2nd, 3rd and 4th iterations respectively */
    $common_bit_mask = ~($ones & $twos);


    /* Remove common bits (the bits
    that appear third time) from 'ones'

    Value of 'ones' will be set as 3,
    0, 0 and 2 after 1st, 2nd, 3rd
    and 4th iterations respectively */
```

```
        $ones &= $common_bit_mask;


        /* Remove common bits (the bits
        that appear third time) from 'twos'

        Value of 'twos' will be set as 0, 3,
        1 and 0 after 1st, 2nd, 3rd and 4th
        itearations respectively */
        $twos &= $common_bit_mask;

        // uncomment this code to see
        // intermediate values
        // printf (" %d %d n", ones, twos);
    }

    return $ones;
}

// Driver Code
$arr = array(3, 3, 2, 3);
$n = sizeof($arr);
echo "The element with single " .
                "occurrence is ",
             getSingle($arr, $n);

// This code is contributed by m_kit
?>
```

Output :

```
The element with single occurrence is 2
```

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

Following is another O(n) time complexity and O(1) extra space method suggested by *aj*. We can sum the bits in same positions for all the numbers and take modulo with 3. The bits for which sum is not multiple of 3, are the bits of number with single occurrence.
Let us consider the example array {5, 5, 5, 8}. The 101, 101, 101, 1000
Sum of first bits%3 = (1 + 1 + 1 + 0)%3 = 0;
Sum of second bits%3 = (0 + 0 + 0 + 0)%0 = 0;
Sum of third bits%3 = (1 + 1 + 1 + 0)%3 = 0;
Sum of fourth bits%3 = (1)%3 = 1;
Hence number which appears once is 1000

C/C++

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

```
#define INT_SIZE 32

int getSingle(int arr[], int n)
{
    // Initialize result
    int result = 0;

    int x, sum;

    // Iterate through every bit
    for (int i = 0; i < INT_SIZE; i++)
    {
      // Find sum of set bits at ith position in all
      // array elements
      sum = 0;
      x = (1 << i);
      for (int j=0; j< n; j++ )
      {
          if (arr[j] & x)
            sum++;
      }

      // The bits with sum not multiple of 3, are the
      // bits of element with single occurrence.
      if (sum % 3)
        result |= x;
    }

    return result;
}

// Driver program to test above function
int main()
{
    int arr[] = {12, 1, 12, 3, 12, 1, 1, 2, 3, 2, 2, 3, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("The element with single occurrence is %d ",
            getSingle(arr, n));
    return 0;
}
```

Java

```
 // JAVA code to find the element
// that occur only once

class GFG
{
```

```
    static final int INT_SIZE = 32;

    // Method to find the element that occur only once
    static int getSingle(int arr[], int n)
    {
        int result = 0;
        int x, sum;

        // Iterate through every bit
        for(int i=0; i<INT_SIZE; i++)
        {
            // Find sum of set bits at ith position in all
            // array elements
            sum = 0;
            x = (1 << i);
            for(int j=0; j<n; j++)
            {
                if((arr[j] & x) == 0)
                sum++;
            }
            // The bits with sum not multiple of 3, are the
            // bits of element with single occurrence.
            if ((sum % 3) == 0)
            result |= x;
        }
        return result;
    }

    // Driver method
    public static void main(String args[])
    {
        int arr[] = {12, 1, 12, 3, 12, 1, 1, 2, 3, 2, 2, 3, 7};
        int n = arr.length;
        System.out.println("The element with single occurrence is " + getSingle(arr, n));
    }

}
// Code contributed by Rishab Jain
```

Python 3

```
 # Python3 code to find the element
# that occur only once
INT_SIZE = 32

def getSingle(arr, n) :

    # Initialize result
```

```
    result = 0

    # Iterate through every bit
    for i in range(0, INT_SIZE) :

        # Find sum of set bits
        # at ith position in all
        # array elements
        sm = 0
        x = (1 << i)
        for j in range(0, n) :
            if (arr[j] & x) :
                sm = sm + 1

        # The bits with sum not
        # multiple of 3, are the
        # bits of element with
        # single occurrence.
        if (sm % 3) :
            result = result | x

    return result

# Driver program
arr = [12, 1, 12, 3, 12, 1, 1, 2, 3, 2, 2, 3, 7]
n = len(arr)
print("The element with single occurrence is "
      ,getSingle(arr, n))


# This code is contributed
# by Nikita Tiwari.
```

C#

```
 // C# code to find the element
// that occur only once
using System;

class GFG
{
    static int INT_SIZE = 32;

    // Method to find the element
    // that occur only once
    static int getSingle(int []arr, int n)
    {
        int result = 0;
```

```
        int x, sum;

        // Iterate through every bit
        for(int i = 0; i < INT_SIZE; i++)
        {
            // Find sum of set bits at ith
            // position in all array elements
            sum = 0;
            x = (1 << i);
            for(int j = 0; j < n; j++)
            {
                if((arr[j] & x) == 0)
                sum++;
            }

            // The bits with sum not multiple
            // of 3, are the bits of element
            // with single occurrence.
            if ((sum % 3) == 0)
            result |= x;
        }
        return result;
    }

    // Driver Code
    public static void Main()
    {
        int []arr = {12, 1, 12, 3, 12, 1,
                     1, 2, 3, 2, 2, 3, 7};
        int n = arr.Length;
        Console.WriteLine("The element with single " +
                "occurrence is " + getSingle(arr, n));
    }

}

// This code is contributed ny vt_m.
```

PHP

```
 <?php
// PHP code to find the element
// that occur only once
$INT_SIZE= 32;

function getSingle($arr, $n)
{
    global $INT_SIZE;
```

```
    // Initialize result
    $result = 0;

    $x; $sum;

    // Iterate through every bit
    for ($i = 0; $i < $INT_SIZE; $i++)
    {
    // Find sum of set bits at ith
    // position in all array elements
    $sum = 0;
    $x = (1 << $i);
    for ($j = 0; $j < $n; $j++ )
    {
        if ($arr[$j] & $x)
            $sum++;
    }

    // The bits with sum not multiple
    // of 3, are the bits of element
    // with single occurrence.
    if ($sum % 3)
        $result |= $x;
    }

    return $result;
}

// Driver Code
$arr = array (12, 1, 12, 3, 12, 1,
              1, 2, 3, 2, 2, 3, 7);
$n = sizeof($arr);
echo "The element with single occurrence is ",
                          getSingle($arr, $n);

// This code is contributed by ajit
?>
```

Output :

```
The element with single occurrence is 7
```

Another approach suggested by *Abhishek Sharma 44*. Add each number once and multiply the sum by 3, we will get thrice the sum of each element of the array. Store it as thrice_sum. Subtract the sum of the whole array from the thrice_sum and divide the result by 2. The number we get is the required number (which appears once in the array).

Array [] : [a, a, a, b, b, b, c, c, c, d]
Mathematical Equation = (3*(a+b+c+d) – (a + a + a + b + b + b + c + c + c + d)) / 2

In more simple words: (3*(sum_of_array_without_duplicates) – (sum_of_array)) / 2

```
let arr[] = {12, 1, 12, 3, 12, 1, 1, 2, 3, 3}
Required no = ( 3*(sum_of_array_without_duplicates) - (sum_of_array) ) / 2
            = ( 3*(12 + 1 + 3 + 2) - (12 + 1 + 12 + 3 + 12 + 1 + 1 + 2 + 3 + 3))/2
            = ( 3*     18          -      50) / 2
            = (54 - 50) / 2
            = 2 (required answer)
```

As we know that **set** does not contain any duplicate element, we will be using set here.

Below is the implementation of above approach:

Python3

```
 # Python program to find the element
# that occur only once

# function which find number
def singleNumber(nums):

    # applying the formula.
    return (3 * sum(set(nums)) - sum(nums)) / 2

# driver function.
a = [12, 1, 12, 3, 12, 1, 1, 2, 3, 2, 2, 3, 7]
print ("The element with single occurrence is ",
                        int(singleNumber(a)))
```

Output :

```
The element with single occurrence is 7
```

This article is compiled by **Sumit Jain** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : jit_t

Source

https://www.geeksforgeeks.org/find-the-element-that-appears-once/

Chapter 162

Find the largest number with n set and m unset bits

Find the largest number with n set and m unset bits - GeeksforGeeks

Given two non-negative numbers **n** and **m**. The problem is to find the largest number having **n** number of set bits and **m** number of unset bits in its binary representation.

Note : 0 bits before leading 1 (or leftmost 1) in binary representation are counted

Contraints: 1 <= n, 0 <= m, (m+n) <= 31

Examples :

```
Input : n = 2, m = 2
Output : 12
(12)10 = (1100)2
We can see that in the binary representation of 12
there are 2 set and 2 unsets bits and it is the largest number.

Input : n = 4, m = 1
Output : 30
```

Following are the steps:

1. Calculate **num** = (1 << (n + m)) – 1. This will produce a number **num** having **(n + m)** number of bits and all are set.
2. Now, toggle the last **m** bits of **num** and then return the toggled number. Refer this post.

C/C++

```
 // C++ implementation to find the largest number
// with n set and m unset bits
#include <bits/stdc++.h>

using namespace std;

// function to toggle the last m bits
unsigned int toggleLastMBits(unsigned int n,
                             unsigned int m)
{
    // if no bits are required to be toggled
    if (m == 0)
        return n;

    // calculating a number 'num' having 'm' bits
    // and all are set
    unsigned int num = (1 << m) - 1;

    // toggle the last m bits and return the number
    return (n ^ num);
}

// function to find the largest number
// with n set and m unset bits
unsigned int largeNumWithNSetAndMUnsetBits(unsigned int n,
                                           unsigned int m)
{
    // calculating a number 'num' having '(n+m)' bits
    // and all are set
    unsigned int num = (1 << (n + m)) - 1;

    // required largest number
    return toggleLastMBits(num, m);
}

// Driver program to test above
int main()
{
    unsigned int n = 2, m = 2;
    cout << largeNumWithNSetAndMUnsetBits(n, m);
    return 0;
}
```

Java

```
 // Java implementation to find the largest number
// with n set and m unset bits
import java.io.*;
```

```
class GFG
{
    // Function to toggle the last m bits
    static int toggleLastMBits(int n, int m)
    {
        // if no bits are required to be toggled
        if (m == 0)
            return n;

        // calculating a number 'num' having 'm' bits
        // and all are set
        int num = (1 << m) - 1;

        // toggle the last m bits and return the number
        return (n ^ num);
    }

    // Function to find the largest number
    // with n set and m unset bits
    static int largeNumWithNSetAndMUnsetBits(int n, int m)
    {
        // calculating a number 'num' having '(n+m)' bits
        // and all are set
        int num = (1 << (n + m)) - 1;

        // required largest number
        return toggleLastMBits(num, m);
    }

    // driver program
    public static void main (String[] args)
    {
        int n = 2, m = 2;
        System.out.println(largeNumWithNSetAndMUnsetBits(n, m));
    }
}

// Contributed by Pramod Kumar
```

Python3

```
 # Python implementation to
# find the largest number
# with n set and m unset bits

# function to toggle
# the last m bits
```

```
def toggleLastMBits(n,m):

    # if no bits are required
    # to be toggled
    if (m == 0):
        return n

    # calculating a number
    # 'num' having 'm' bits
    # and all are set
    num = (1 << m) - 1

    # toggle the last m bits
    # and return the number
    return (n ^ num)


# function to find
# the largest number
# with n set and m unset bits
def largeNumWithNSetAndMUnsetBits(n,m):

    # calculating a number
    # 'num' having '(n+m)' bits
    # and all are set
    num = (1 << (n + m)) - 1

    # required largest number
    return toggleLastMBits(num, m)

# Driver code

n = 2
m = 2

print(largeNumWithNSetAndMUnsetBits(n, m))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# implementation to find the largest number
// with n set and m unset bits
using System;

class GFG
{
```

```
    // Function to toggle the last m bits
    static int toggleLastMBits(int n, int m)
    {
        // if no bits are required to be toggled
        if (m == 0)
            return n;

        // calculating a number 'num' having 'm' bits
        // and all are set
        int num = (1 << m) - 1;

        // toggle the last m bits and return the number
        return (n ^ num);
    }

    // Function to find the largest number
    // with n set and m unset bits
    static int largeNumWithNSetAndMUnsetBits(int n, int m)
    {
        // calculating a number 'num' having '(n+m)' bits
        // and all are set
        int num = (1 << (n + m)) - 1;

        // required largest number
        return toggleLastMBits(num, m);
    }

    // Driver program
    public static void Main ()
    {
        int n = 2, m = 2;
        Console.Write(largeNumWithNSetAndMUnsetBits(n, m));
    }

}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP implementation to find
// the largest number with n
// set and m unset bits

// function to toggle
// the last m bits
function toggleLastMBits($n, $m)
```

```
{
    // if no bits are required
    // to be toggled
    if ($m == 0)
        return $n;

    // calculating a number 'num'
    // having 'm' bits and all are set
    $num = (1 << $m) - 1;

    // toggle the last m bits
    // and return the number
    return ($n ^ $num);
}

// function to find the largest number
// with n set and m unset bits
function largeNumWithNSetAndMUnsetBits($n,
                                       $m)
{
    // calculating a number 'num'
    // having '(n+m)' bits and all are set
    $num = (1 << ($n + $m)) - 1;

    // required largest number
    return toggleLastMBits($num, $m);
}

// Driver Code
$n = 2; $m = 2;
echo largeNumWithNSetAndMUnsetBits($n, $m);

// This code is contributed by vt_m.
?>
```

Output :

```
12
```

For greater values of **n** and **m**, you can use **long int** and **long long int** datatypes to generate the required number.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/find-largest-number-n-set-m-unset-bits/

Chapter 163

Find the maximum subarray XOR in a given array

Find the maximum subarray XOR in a given array - GeeksforGeeks

Given an array of integers. find the maximum XOR subarray value in given array. Expected time complexity O(n).

Examples:

```
Input: arr[] = {1, 2, 3, 4}
Output: 7
The subarray {3, 4} has maximum XOR value

Input: arr[] = {8, 1, 2, 12, 7, 6}
Output: 15
The subarray {1, 2, 12} has maximum XOR value

Input: arr[] = {4, 6}
Output: 6
The subarray {6} has maximum XOR value
```

A **Simple Solution** is to use two loops to find XOR of all subarrays and return the maximum.

C++

```
 // A simple C++ program to find max subarray XOR
#include<bits/stdc++.h>
using namespace std;

int maxSubarrayXOR(int arr[], int n)
```

```
{
    int ans = INT_MIN;     // Initialize result

    // Pick starting points of subarrays
    for (int i=0; i<n; i++)
    {
        int curr_xor = 0; // to store xor of current subarray

        // Pick ending points of subarrays starting with i
        for (int j=i; j<n; j++)
        {
            curr_xor = curr_xor ^ arr[j];
            ans = max(ans, curr_xor);
        }
    }
    return ans;
}

// Driver program to test above functions
int main()
{
    int arr[] = {8, 1, 2, 12};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Max subarray XOR is " << maxSubarrayXOR(arr, n);
    return 0;
}
```

Java

```
 // A simple Java program to find max subarray XOR
class GFG {
    static int maxSubarrayXOR(int arr[], int n)
    {
        int ans = Integer.MIN_VALUE; // Initialize result

        // Pick starting points of subarrays
        for (int i=0; i<n; i++)
        {
                // to store xor of current subarray
            int curr_xor = 0;

            // Pick ending points of subarrays starting with i
            for (int j=i; j<n; j++)
            {
                curr_xor = curr_xor ^ arr[j];
                ans = Math.max(ans, curr_xor);
            }
        }
```

```
        return ans;
    }

    // Driver program to test above functions
    public static void main(String args[])
    {
        int arr[] = {8, 1, 2, 12};
        int n = arr.length;
        System.out.println("Max subarray XOR is " +
                                maxSubarrayXOR(arr, n));
    }
}
//This code is contributed by Sumit Ghosh
```

Python3

```
 # A simple Python program
# to find max subarray XOR

def maxSubarrayXOR(arr,n):

    ans = -2147483648     #Initialize result

    # Pick starting points of subarrays
    for i in range(n):

        # to store xor of current subarray
        curr_xor = 0

        # Pick ending points of
        # subarrays starting with i
        for j in range(i,n):

            curr_xor = curr_xor ^ arr[j]
            ans = max(ans, curr_xor)


    return ans


# Driver code

arr = [8, 1, 2, 12]
n = len(arr)

print("Max subarray XOR is ",
     maxSubarrayXOR(arr, n))
```

```
# This code is contributed
# by Anant Agarwal.
```

C#

```
 // A simple C# program to find
// max subarray XOR
using System;

class GFG
{

    // Function to find max subarray
    static int maxSubarrayXOR(int []arr, int n)
    {
        int ans = int.MinValue;
        // Initialize result

        // Pick starting points of subarrays
        for (int i = 0; i < n; i++)
        {
            // to store xor of current subarray
            int curr_xor = 0;

            // Pick ending points of
            // subarrays starting with i
            for (int j = i; j < n; j++)
            {
                curr_xor = curr_xor ^ arr[j];
                ans = Math.Max(ans, curr_xor);
            }
        }
        return ans;
    }

    // Driver code
    public static void Main()
    {
        int []arr = {8, 1, 2, 12};
        int n = arr.Length;
        Console.WriteLine("Max subarray XOR is " +
                           maxSubarrayXOR(arr, n));
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// A simple PHP program to
// find max subarray XOR

function maxSubarrayXOR($arr, $n)
{

    // Initialize result
    $ans = PHP_INT_MIN;

    // Pick starting points
    // of subarrays
    for ($i = 0; $i < $n; $i++)
    {
        // to store xor of
        // current subarray
        $curr_xor = 0;

        // Pick ending points of
        // subarrays starting with i
        for ($j = $i; $j < $n; $j++)
        {
            $curr_xor = $curr_xor ^ $arr[$j];
            $ans = max($ans, $curr_xor);
        }
    }
    return $ans;
}

    // Driver Code
    $arr = array(8, 1, 2, 12);
    $n = count($arr);
    echo "Max subarray XOR is "
         , maxSubarrayXOR($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
Max subarray XOR is 15
```

Time Complexity of above solution is $O(n^2)$.

An **Efficient Solution** can solve the above problem in O(n) time under the assumption that integers take fixed number of bits to store. The idea is to use Trie Data Structure. Below is algorithm.

```
1) Create an empty Trie.  Every node of Trie is going to
   contain two children, for 0 and 1 value of bit.
2) Initialize pre_xor = 0 and insert into the Trie.
3) Initialize result = minus infinite
4) Traverse the given array and do following for every
   array element arr[i].
       a) pre_xor  = pre_xor  ^ arr[i]
          pre_xor now contains xor of elements from
          arr[0] to arr[i].
       b) Query the maximum xor value ending with arr[i]
          from Trie.
       c) Update result if the value obtained in step
          4.b is more than current value of result.
```

How does 4.b work?
We can observe from above algorithm that we build a Trie that contains XOR of all prefixes of given array. To find the maximum XOR subarray ending with arr[i], there may be two cases.
i) The prefix itself has the maximum XOR value ending with arr[i]. For example if i=2 in {8, 2, 1, 12}, then the maximum subarray xor ending with arr[2] is the whole prefix.
ii) We need to remove some prefix (ending at index from 0 to i-1). For example if i=3 in {8, 2, 1, 12}, then the maximum subarray xor ending with arr[3] starts with arr[1] and we need to remove arr[0].

To find the prefix to be removed, we find the entry in Trie that has maximum XOR value with current prefix. If we do XOR of such previous prefix with current prefix, we get the maximum XOR value ending with arr[i].
If there is no prefix to be removed (case i), then we return 0 (that's why we inserted 0 in Trie).

Below is the implementation of above idea :

C++

```
 // C++ program for a Trie based O(n) solution to find max
// subarray XOR
#include<bits/stdc++.h>
using namespace std;

// Assumed int size
#define INT_SIZE 32

// A Trie Node
struct TrieNode
{
    int value;  // Only used in leaf nodes
    TrieNode *arr[2];
};
```

```
// Utility function tp create a Trie node
TrieNode *newNode()
{
    TrieNode *temp = new TrieNode;
    temp->value = 0;
    temp->arr[0] = temp->arr[1] = NULL;
    return temp;
}

// Inserts pre_xor to trie with given root
void insert(TrieNode *root, int pre_xor)
{
    TrieNode *temp = root;

    // Start from the msb, insert all bits of
    // pre_xor into Trie
    for (int i=INT_SIZE-1; i>=0; i--)
    {
        // Find current bit in given prefix
        bool val = pre_xor & (1<<i);

        // Create a new node if needed
        if (temp->arr[val] == NULL)
            temp->arr[val] = newNode();

        temp = temp->arr[val];
    }

    // Store value at leaf node
    temp->value = pre_xor;
}

// Finds the maximum XOR ending with last number in
// prefix XOR 'pre_xor' and returns the XOR of this maximum
// with pre_xor which is maximum XOR ending with last element
// of pre_xor.
int query(TrieNode *root, int pre_xor)
{
    TrieNode *temp = root;
    for (int i=INT_SIZE-1; i>=0; i--)
    {
        // Find current bit in given prefix
        bool val = pre_xor & (1<<i);

        // Traverse Trie, first look for a
        // prefix that has opposite bit
        if (temp->arr[1-val]!=NULL)
```

```
            temp = temp->arr[1-val];

        // If there is no prefix with opposite
        // bit, then look for same bit.
        else if (temp->arr[val] != NULL)
            temp = temp->arr[val];
    }
    return pre_xor^(temp->value);
}

// Returns maximum XOR value of a subarray in arr[0..n-1]
int maxSubarrayXOR(int arr[], int n)
{
    // Create a Trie and insert 0 into it
    TrieNode *root = newNode();
    insert(root, 0);

    // Initialize answer and xor of current prefix
    int result = INT_MIN, pre_xor =0;

    // Traverse all input array element
    for (int i=0; i<n; i++)
    {
        // update current prefix xor and insert it into Trie
        pre_xor = pre_xor^arr[i];
        insert(root, pre_xor);

        // Query for current prefix xor in Trie and update
        // result if required
        result = max(result, query(root, pre_xor));
    }
    return result;
}

// Driver program to test above functions
int main()
{
    int arr[] = {8, 1, 2, 12};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Max subarray XOR is " << maxSubarrayXOR(arr, n);
    return 0;
}
```

Java

```
 // Java program for a Trie based O(n) solution to
// find max subarray XOR
class GFG
```

```
{
    // Assumed int size
    static final int INT_SIZE = 32;

    // A Trie Node
    static class TrieNode
    {
        int value;  // Only used in leaf nodes
        TrieNode[] arr =  new TrieNode[2];
        public TrieNode() {
            value = 0;
            arr[0] = null;
            arr[1] = null;
        }
    }
    static TrieNode root;

    // Inserts pre_xor to trie with given root
    static void insert(int pre_xor)
    {
        TrieNode temp = root;

        // Start from the msb, insert all bits of
        // pre_xor into Trie
        for (int i=INT_SIZE-1; i>=0; i--)
        {
            // Find current bit in given prefix
            int val = (pre_xor & (1<<i)) >=1 ? 1 : 0;

            // Create a new node if needed
            if (temp.arr[val] == null)
                temp.arr[val] = new TrieNode();

            temp = temp.arr[val];
        }

        // Store value at leaf node
        temp.value = pre_xor;
    }

    // Finds the maximum XOR ending with last number in
    // prefix XOR 'pre_xor' and returns the XOR of this
    // maximum with pre_xor which is maximum XOR ending
    // with last element of pre_xor.
    static int query(int pre_xor)
    {
        TrieNode temp = root;
        for (int i=INT_SIZE-1; i>=0; i--)
```

```
    {
        // Find current bit in given prefix
        int val = (pre_xor & (1<<i)) >= 1 ? 1 : 0;

        // Traverse Trie, first look for a
        // prefix that has opposite bit
        if (temp.arr[1-val] != null)
            temp = temp.arr[1-val];

        // If there is no prefix with opposite
        // bit, then look for same bit.
        else if (temp.arr[val] != null)
            temp = temp.arr[val];
    }
    return pre_xor^(temp.value);
}

// Returns maximum XOR value of a subarray in
    // arr[0..n-1]
static int maxSubarrayXOR(int arr[], int n)
{
    // Create a Trie and insert 0 into it
    root = new TrieNode();
    insert(0);

    // Initialize answer and xor of current prefix
    int result = Integer.MIN_VALUE;
    int pre_xor =0;

    // Traverse all input array element
    for (int i=0; i<n; i++)
    {
        // update current prefix xor and insert it
            // into Trie
        pre_xor = pre_xor^arr[i];
        insert(pre_xor);

        // Query for current prefix xor in Trie and
        // update result if required
        result = Math.max(result, query(pre_xor));

    }
    return result;
}

// Driver program to test above functions
public static void main(String args[])
{
```

```
        int arr[] = {8, 1, 2, 12};
        int n = arr.length;
        System.out.println("Max subarray XOR is " +
                                    maxSubarrayXOR(arr, n));
    }
}
// This code is contributed by Sumit Ghosh
```

Output:

```
Max subarray XOR is 15
```

Exercise: Extend the above solution so that it also prints starting and ending indexes of subarray with maximum value (Hint: we can add one more field to Trie node to achieve this)

This article is contributed by **Romil Punetha**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : Sam007, vt_m

Source

https://www.geeksforgeeks.org/find-the-maximum-subarray-xor-in-a-given-array/

Chapter 164

Find the maximum subset XOR of a given set

Find the maximum subset XOR of a given set - GeeksforGeeks

Given an set of positive integers. find the maximum XOR subset value in the given set. Expected time complexity O(n).

Examples:

```
Input: set[] = {2, 4, 5}
Output: 7
The subset {2, 5} has maximum XOR value

Input: set[] = {9, 8, 5}
Output: 13
The subset {8, 5} has maximum XOR value

Input: set[] = {8, 1, 2, 12, 7, 6}
Output: 15
The subset {1, 2, 12} has maximum XOR value

Input: set[] = {4, 6}
Output: 6
The subset {6} has maximum XOR value
```

Note that this problem is different from the maximum subarray XOR. Here we need to find subset instead of subarray.

A **Simple Solution** is to generate all possible subsets of given set, find XOR of every subset and return the subset with maximum XOR.

Below is an **Efficient Algorithm** that works in O(n) time.
The idea is based on below facts:

1. Number of bits to represent all elements is fixed which is 32 bits for integer in most of the compilers.
2. If maximum element has Most Significant Bit MSB at position i, then result is at least 2^i

```
1. Initialize index of chosen elements as 0. Let this index be
   'index'
2. Traverse through all bits starting from most significant bit.
   Let i be the current bit.
......(a) Find the maximum element with i'th bit set.  If there
          is no element with i'th bit set, continue to smaller
          bit.
......(b) Let the element with i'th bit set be maxEle and index
          of this element be maxInd. Place maxEle at 'index' and
          (by swapping set[index] and set[maxInd])
......(c) Do XOR of maxEle with all numbers having i'th  bit as set.
......(d) Increment index
3. Return XOR of all elements in set[]. Note that set[] is modified
   in step 2.c.
```

Below is the implementation of this algorithm.

C++

```
 // C++ program to find
// maximum XOR subset
#include<bits/stdc++.h>
using namespace std;

// Number of bits to
// represent int
#define INT_BITS 32

// Function to return
// maximum XOR subset
// in set[]
int maxSubarrayXOR(int set[], int n)
{
    // Initialize index of
    // chosen elements
    int index = 0;

    // Traverse through all
    // bits of integer
    // starting from the most
    // significant bit (MSB)
    for (int i = INT_BITS-1; i >= 0; i--)
```

```
{
    // Initialize index of
    // maximum element and
    // the maximum element
    int maxInd = index;
    int maxEle = INT_MIN;
    for (int j = index; j < n; j++)
    {
        // If i'th bit of set[j]
        // is set and set[j] is
        // greater than max so far.
        if ( (set[j] & (1 << i)) != 0
                 && set[j] > maxEle )
            maxEle = set[j], maxInd = j;
    }

    // If there was no
    // element with i'th
    // bit set, move to
    // smaller i
    if (maxEle == INT_MIN)
    continue;

    // Put maximum element
    // with i'th bit set
    // at index 'index'
    swap(set[index], set[maxInd]);

    // Update maxInd and
    // increment index
    maxInd = index;

    // Do XOR of set[maxIndex]
    // with all numbers having
    // i'th bit as set.
    for (int j=0; j<n; j++)
    {
        // XOR set[maxInd] those
        // numbers which have the
        // i'th bit set
        if (j != maxInd &&
           (set[j] & (1 << i)) != 0)
            set[j] = set[j] ^ set[maxInd];
    }

    // Increment index of
    // chosen elements
    index++;
```

```
    }

    // Final result is
    // XOR of all elements
    int res = 0;
    for (int i = 0; i < n; i++)
        res ^= set[i];
    return res;
}

// Driver program
int main()
{
    int set[] = {9, 8, 5};
    int n = sizeof(set) / sizeof(set[0]);

    cout << "Max subset XOR is ";
    cout << maxSubarrayXOR(set, n);
    return 0;
}
```

Java

```
 // Java program to find
// maximum XOR subset
import java.util.*;

class GFG {

// Number of bits to
// represent int
static final int INT_BITS = 32;

// Function to return
// maximum XOR subset
// in set[]
static int maxSubarrayXOR(int set[], int n)
{
    // Initialize index of
    // chosen elements
    int index = 0;

    // Traverse through all
    // bits of integer
    // starting from the most
    // significant bit (MSB)
    for (int i = INT_BITS - 1; i >= 0; i--)
    {
```

```
// Initialize index of
// maximum element and
// the maximum element
int maxInd = index;
int maxEle = Integer.MIN_VALUE;
for (int j = index; j < n; j++) {

    // If i'th bit of set[j]
    // is set and set[j] is
    // greater than max so far.
    if ((set[j] & (1 << i)) != 0 && set[j] > maxEle)
    {
    maxEle = set[j];
    maxInd = j;
    }
}

// If there was no
// element with i'th
// bit set, move to
// smaller i
if (maxEle == -2147483648)
    continue;

// Put maximum element
// with i'th bit set
// at index 'index'
int temp = set[index];
set[index] = set[maxInd];
set[maxInd] = temp;

// Update maxInd and
// increment index
maxInd = index;

// Do XOR of set[maxIndex]
// with all numbers having
// i'th bit as set.
for (int j = 0; j < n; j++) {

    // XOR set[maxInd] those
    // numbers which have the
    // i'th bit set
    if (j != maxInd && (set[j] & (1 << i)) != 0)
    set[j] = set[j] ^ set[maxInd];
}

// Increment index of
```

```
    // chosen elements
    index++;
    }

    // Final result is
    // XOR of all elements
    int res = 0;
    for (int i = 0; i < n; i++)
    res ^= set[i];
    return res;
}

// Driver code
public static void main(String arg[]) {
    int set[] = {9, 8, 5};
    int n = set.length;

    System.out.print("Max subset XOR is ");
    System.out.print(maxSubarrayXOR(set, n));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python program to find
# maximum XOR subset

# Number of bits to
# represent int
INT_BITS=32

# Function to return
# maximum XOR subset
# in set[]
def maxSubarrayXOR(set,n):

    # Initialize index of
    # chosen elements
    index = 0

    # Traverse through all
    # bits of integer
    # starting from the most
    # significant bit (MSB)
    for i in range(INT_BITS-1,-1,-1):
```

```
# Initialize index of
# maximum element and
# the maximum element
maxInd = index
maxEle = -2147483648
for j in range(index,n):

    # If i'th bit of set[j]
    # is set and set[j] is
    # greater than max so far.
    if ( (set[j] & (1 << i)) != 0
              and set[j] > maxEle ):

        maxEle = set[j]
        maxInd = j

# If there was no
# element with i'th
# bit set, move to
# smaller i
if (maxEle ==-2147483648):
    continue

# Put maximum element
# with i'th bit set
# at index 'index'
temp=set[index]
set[index]=set[maxInd]
set[maxInd]=temp

# Update maxInd and
# increment index
maxInd = index

# Do XOR of set[maxIndex]
# with all numbers having
# i'th bit as set.
for j in range(n):

    # XOR set[maxInd] those
    # numbers which have the
    # i'th bit set
    if (j != maxInd and
       (set[j] & (1 << i)) != 0):
        set[j] = set[j] ^ set[maxInd]


# Increment index of
```

```
        # chosen elements
        index=index + 1


    # Final result is
    # XOR of all elements
    res = 0
    for i in range(n):
        res =res ^ set[i]
    return res

# Driver code

set= [9, 8, 5]
n =len(set)

print("Max subset XOR is ",end="")
print(maxSubarrayXOR(set, n))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to find
// maximum XOR subset
using System;

class GFG
{

    // Number of bits to
    // represent int
    static int INT_BITS = 32;

    // Function to return
    // maximum XOR subset
    // in set[]
    static int maxSubarrayXOR(int []set,
                              int n)
    {

        // Initialize index of
        // chosen elements
        int index = 0;

        // Traverse through all
        // bits of integer
```

```
// starting from the most
// significant bit (MSB)
for (int i = INT_BITS - 1;
         i >= 0; i--)
{

// Initialize index of
// maximum element and
// the maximum element
int maxInd = index;
int maxEle = int.MinValue;
for (int j = index; j < n; j++)
{

    // If i'th bit of set[j]
    // is set and set[j] is
    // greater than max so far.
    if ((set[j] & (1 << i)) != 0 &&
         set[j] > maxEle)
    {
        maxEle = set[j];
        maxInd = j;
    }
}

// If there was no
// element with i'th
// bit set, move to
// smaller i
if (maxEle == -2147483648)
    continue;

// Put maximum element
// with i'th bit set
// at index 'index'
int temp = set[index];
set[index] = set[maxInd];
set[maxInd] = temp;

// Update maxInd and
// increment index
maxInd = index;

// Do XOR of set[maxIndex]
// with all numbers having
// i'th bit as set.
for (int j = 0; j < n; j++)
{
```

```
            // XOR set[maxInd] those
            // numbers which have the
            // i'th bit set
            if (j != maxInd && (set[j] &
               (1 << i)) != 0)
            set[j] = set[j] ^ set[maxInd];
    }

        // Increment index of
        // chosen elements
        index++;
    }

    // Final result is
    // XOR of all elements
        int res = 0;
        for (int i = 0; i < n; i++)
        res ^= set[i];
        return res;
    }

    // Driver code
    public static void Main()
    {
        int []set = {9, 8, 5};
        int n = set.Length;

        Console.Write("Max subset XOR is ");
        Console.Write(maxSubarrayXOR(set, n));
    }
}

// This code is contributed by Sam007.
```

Output:

```
Max subset XOR is 13
```

Illustration:

```
Let the input set be : {9, 8, 5}

We start from 31st bit [Assuming Integers are 32 bit
long]. The loop will continue without doing anything till 4'th bit.
```

```
The 4th bit is set in set[0] i.e. 9 and this is the maximum
element with 4th bit set. So we choose this element and check
if any other number has the same bit set. If yes, we XOR that
number with 9. The element set[1], i.e., 8 also has 4'th bit
set. Now set[] becomes {9, 1, 5}.  We add 9 to the list of
chosen elements by incrementing 'index'

We move further and find the maximum number with 3rd bit set
which is set[2] i.e. 5  No other number in the array has 3rd
bit set. 5 is also added to the list of chosen element.

We then iterate for bit 2 (no number for this) and then for
1 which is 1. But numbers 9 and 5 have the 1st bit set. Thus
we XOR 9 and 5 with 1 and our set becomes (8, 1, 4)

Finally, we XOR current elements of set and get the result
as 8 ^ 1 ^ 4 = 13.
```

How does this work?
Let us first understand a simple case when all elements have Most Significant Bits (MSBs) at different positions. The task in this particular case is simple, we need to do XOR of all elements.
If input contains multiple numbers with the same MSB, then it's not obvious which of them we should choose to include in the XOR. What we do is reduce the input list into an equivalent form that doesn't contain more than one number of the same length. By taking the maximum element, we know that the MSB of this is going to be there in output. Let this MSB be at position i. If there are more elements with i'th set (or same MSB), we XOR them with the maximum number so that the i'th bit becomes 0 in them and problem reduces to i-1 bits.

References:
http://math.stackexchange.com/questions/48682/maximization-with-xor-operator

Note: The above method is similar to Gaussian elimination. Consider a matrix of size 32 x n where 32 is number of bits and n is number of elements in array.

This article is contributed by Ekta Goel. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : Sam007

Source

https://www.geeksforgeeks.org/find-maximum-subset-xor-given-set/

Chapter 165

Find the missing element in an array of integers represented in binary format

Find the missing element in an array of integers represented in binary format - GeeksforGeeks

Given N strings which represents all integers from 0 to N in binary format except any one. The task is to find the missing number. Input consists of an array of strings where array elements are represented in binary format.

Examples:

> **Input:** arr[] = {“0000”, “0001”, “0010”, “0100”}
> **Output:** 3
>
> **Input:** arr[] = {“0000”, “0001”, “0010”, “0011”, “0100”, “0110”, “0111”, “1000”}
> **Output**: 5

Approach:

- An imbalance of 1’s and 0’s in the least significant bits of the numbers can be observed in the N integers given. Since one number is missing either a 0 or 1 from the LSB is missing. If the number which is missing has LSB = 0 then count(1) will be greater than equal to count(0). If LSB of missing number is 1 then count(1) is less than count(0).
- From the step 1 one can easily determine the LSB of missing number.
- Once determined, discard all the numbers having LSB different from that of the missing number, i.e., if the missing number has LSB = 0, then discard all the numbers with LSB = 1 and vice versa.
- Continue the process from step 1 all over again and recur for the next LSB.

- Continue with the above process till all the bits are traversed.

Below is the impplementation of the above approach:

```
 // C++ program to find the missing integer
// in N numbers when N bits are given
#include <bits/stdc++.h>
using namespace std;

class BitInteger {
private:
    bool* bits;

public:
    static const int INTEGER_SIZE = 32;

    BitInteger()
    {
        bits = new bool[INTEGER_SIZE];
    }

    // Constructor to convert an integer
    // variable into binary format
    BitInteger(int value)
    {
        bits = new bool[INTEGER_SIZE];

        for (int j = 0; j < INTEGER_SIZE; j++) {

            // The if statement will shift the
            // original value j times.
            // So that appropriate (INTEGER_SIZE - 1 -j)th
            // bits will be either 0/1.
            //  (INTEGER_SIZE - 1 -j)th bit for all
            // j = 0 to INTEGER_SIZE-1 corresponds
            // to  LSB to MSB respectively.
            if (((value >> j) & 1) == 1)
                bits[INTEGER_SIZE - 1 - j] = true;
            else
                bits[INTEGER_SIZE - 1 - j] = false;
        }
    }
    // Constructor to convert a
    // string into binary format.
    BitInteger(string str)
    {
        int len = str.length();
        int x = INTEGER_SIZE - len;
```

```
        bits = new bool[INTEGER_SIZE];

        // If len = 4. Then x = 32 - 4 = 28.
        // Hence iterate from
        // bit 28 to bit 32 and just
        // replicate the input string.
        int i = 0;

        for (int j = x; j <= INTEGER_SIZE && i < len; j++, i++) {
            if (str[i] == '1')
                bits[j] = true;
            else
                bits[j] = false;
        }
    }

    // this function fetches the kth bit
    int fetch(int k)
    {
        if (bits[k])
            return 1;

        return 0;
    }

    // this function will set a value
    // of bit indicated by k to given bitValue
    void set(int k, int bitValue)
    {
        if (bitValue == 0)
            bits[k] = false;
        else
            bits[k] = true;
    }

    // convert binary representation to integer
    int toInt()
    {
        int n = 0;
        for (int i = 0; i < INTEGER_SIZE; i++) {
            n = n << 1;
            if (bits[i])
                n = n | 1;
        }
        return n;
    }
};
```

```
// Function to find the missing number
int findMissingFunc(list<BitInteger>& myList, int column)
{
    // This means that we have processed
    // the entire 32 bit binary number.
    if (column < 0)
        return 0;

    list<BitInteger> oddIndices;
    list<BitInteger> evenIndices;

    for (BitInteger t : myList) {

        // Initially column = LSB. So
        // if LSB of the given number is 0,
        // then the number is even and
        // hence we add it to evenIndices list.
        // else if LSB = 0 then add it to oddIndices list.
        if (t.fetch(column) == 0)
            evenIndices.push_back(t);
        else
            oddIndices.push_back(t);
    }

    // Step 1 and Step 2 of the algorithm.
    // Here we determine the LSB bit of missing number.

    if (oddIndices.size() >= evenIndices.size())

        // LSB of the missing number is 0.
        // Hence it is an even number.
        // Step 3 and 4 of the algorithm
        // (discarding all odd numbers)
        return (findMissingFunc(evenIndices, column - 1)) << 1 | 0;

    else
        // LSB of the missing number is 1.
        // Hence it is an odd number.
        // Step 3 and 4 of the algorithm
        // (discarding all even numbers)
        return (findMissingFunc(oddIndices, column - 1)) << 1 | 1;
}

// Function to return the missing integer
int findMissing(list<BitInteger>& myList)
{
    // Initial call is with given array and LSB.
    return findMissingFunc(myList, BitInteger::INTEGER_SIZE - 1);
```

```
}

// Driver Code.
int main()
{

    // a corresponds to the input array which
    // is a list of binary numbers
    list<BitInteger> a = { BitInteger("0000"), BitInteger("0001"),
                           BitInteger("0010"), BitInteger("0100"),
                           BitInteger("0101") };
    int missing1 = findMissing(a);
    cout << missing1 << "\n";

    return 0;
}
```

Output:

```
3
```

Time Complexity: O(N)

Source

https://www.geeksforgeeks.org/find-the-missing-element-in-an-array-of-integers-represented-in-binary-format/

Chapter 166

Find the n-th number whose binary representation is a palindrome

Find the n-th number whose binary representation is a palindrome - GeeksforGeeks

Find the nth number whose binary representation is a palindrome. Do not consider the leading zeros, while considering the binary representation. Consider the 1st number whose binary representation is palindrome as 1, instead of 0

Examples:

```
Input : 1
Output : 1
1st Number whose binary representation
is palindrome is 1 (1)

Input : 9
Output : 27
9th Number whose binary representation
is palindrome is 27 (11011)
```

Method 1: Naive

Naive approach would be, traverse through all the integers from 1 to 2^31 – 1 and increment palindrome count, if the number is palindrome. When the palindrome count reaches the required n, break the loop and return the current integer.

C

```
 // C program to find n-th number whose binary
// representation is palindrome.
#include <stdio.h>
#define INT_MAX 2147483647

/* Finds if the kth bit is set in the binary
   representation */
int isKthBitSet(int x, int k)
{
    return (x & (1 << (k - 1))) ? 1 : 0;
}

/* Returns the position of leftmost set bit
   in the binary representation */
int leftmostSetBit(int x)
{
    int count = 0;
    while (x) {
        count++;
        x = x >> 1;
    }
    return count;
}

/* Finds whether the integer in binary
   representation is palindrome or not*/
int isBinPalindrome(int x)
{
    int l = leftmostSetBit(x);
    int r = 1;

    // One by one compare bits
    while (l > r) {

        // Compare left and right bits and converge
        if (isKthBitSet(x, l) != isKthBitSet(x, r))
            return 0;
        l--;
        r++;
    }
    return 1;
}

int findNthPalindrome(int n)
{
    int pal_count = 0;

    /* Start from 1, traverse through
```

```
      all the integers */
    int i = 0;
    for (i = 1; i <= INT_MAX; i++) {
        if (isBinPalindrome(i)) {
            pal_count++;
        }
        /* If we reach n, break the loop */
        if (pal_count == n)
            break;
    }

    return i;
}

// Driver code
int main()
{
    int n = 9;
    printf("%d", findNthPalindrome(n));
}
```

Java

```
 // Java program to find n-th
// number whose binary
// representation is palindrome.
import java.io.*;

class GFG
{
static int INT_MAX = 2147483647;

/* Finds if the kth bit
is set in the binary
representation */
static int isKthBitSet(int x, int k)
{
    return ((x & (1 <<
            (k - 1))) > 0) ? 1 : 0;
}

/* Returns the position of
leftmost set bit in the
binary representation */
static int leftmostSetBit(int x)
{
    int count = 0;
    while (x > 0)
```

```
    {
        count++;
        x = x >> 1;
    }
    return count;
}

/* Finds whether the integer
in binary representation is
palindrome or not*/
static int isBinPalindrome(int x)
{
    int l = leftmostSetBit(x);
    int r = 1;

    // One by one compare bits
    while (l > r)
    {

        // Compare left and right
        // bits and converge
        if (isKthBitSet(x, l) !=
            isKthBitSet(x, r))
            return 0;
        l--;
        r++;
    }
    return 1;
}

static int findNthPalindrome(int n)
{
    int pal_count = 0;

    /* Start from 1, traverse
    through all the integers */
    int i = 0;
    for (i = 1; i <= INT_MAX; i++)
    {
        if (isBinPalindrome(i) > 0)
        {
            pal_count++;
        }

        /* If we reach n,
        break the loop */
        if (pal_count == n)
            break;
```

```
    }

    return i;
}

// Driver code
public static void main (String[] args)
{
    int n = 9;
    System.out.println(findNthPalindrome(n));
}
}

// This code is contributed
// by anuj_67.
```

```
27
```

Time complexity of this solution is **O(x)** where x is resultant number. Note that value of x is generally much larger than n.

Method 2: Constructing the nth palindrome

We can construct the nth binary palindrome in its binary representation directly using the below approach.
If we observe first few binary palindromes

```
 *         | nth Binary  |
       n   | Palindrome  |     Group
           |             |
--------------------------- Group 0
    1 --->  1 (1)


Group 1 (Will have binary representation of length 2*(1)
and 2*(1) + 1)

    Fix the first and last bit as 1 and insert nothing
    (|) in between. Length is 2*(1)
    2 --->  1|1 (3)

    Fix the first and last bit as 1 and insert bit 0
    in between. Length is 2*(1) + 1
    3 --->  101 (5)

    Fix the first and last bit as 1 and insert bit 1
    in between. Length is 2*(1) + 1
```

```
    4 --->  111 (7)
    F

Group 2 (Will have binary representation of length
    2*(2) and 2*(2) + 1).  Fix the first and last
    bit as 1 and insert nothing (|) at middle.
    And put 0 in binary format in both directions
    from middle. Length is 2*(2)
    5 --->  10|01
    Fix the first and last bit as 1 and insert
    nothing (|) at middle. And put 1 in binary
    format in both directions from middle.
    Length is 2*(2)
    6 --->  11|11

    7 --->  10001
    8 --->  10101
    9 --->  11011
    10 --->  11111

Group 3 (Will have binary representation of
        length 2*(3) and 2*(3) + 1)
    11 ---> 100|001
    12 ---> 101|101
    13 ---> 110|011
    14 ---> 111|111

    15 ---> 1000001
    16 ---> 1001001
    17 ---> 1010101
    18 ---> 1011101
    19 ---> 1100011
    20 ---> 1101011
    21 ---> 1110111
    22 ---> 1111111
--------------------
```

Algorithm:
1) We can divide the set of palindrome numbers into some groups.
2) n-th group will have (2^(n-1) + 2^n = 3 * 2 ^(n-1)) number of binary palindromes
3) With the given number, we can find the group to which it belongs to and the offset in that group.
4) As the leading zeros are not to be considered, we should use bit 1 as the starting bit and ending bit of the number in binary representation
5) And we will fill other bits based on the groupno and groupoffset
6) Based on the offset, we can find which bit should be inserted at the middle (|(nothing) or 0 or 1) and
which number(in binary form) (1 or 2 or 3 or 4 or ..) should be placed in both directions

from middle

Consider Below Example

```
Let us construct the 8th binary palindrome number
The group number will be 2, and no.of elements
before that group are 1 + 3 * 2^1 which is 4

So the offset for the 8th element will be 8 - 4
 - 1 = 3

And first 2^(groupno - 1) = 2^1, elements will
have even length(in binary representation) of
2*groupno, next 2^groupno elements will have odd
length(in binary representation) of 2*groupno + 1

Place bit 1 as the first bit and as the last bit
(firstbit: 0, last bit: 2*groupno or 2*groupno - 1)

As the offset is 3, 4th(3 + 1) element in the
group, will have odd length and have 1 in the
middle

Below is the table of middle bit to be used for
the given offset for the group 2
offset    middle bit
  0            |
  1            |
  2            0
  3            1
  4            0
  5            1
And we should be filling the binary representation
of number 0(((groupoffset) - 2^(groupno-1)) /2)
from middle n both directions
1 0 1 0 1
FirstElement Number MiddleElement Number LastElement
    1           0         1         0         1

The 8th number will be 21
```

C

```
 // Efficient C program to find n-th palindrome
#include <stdio.h>
#define INT_SIZE 32
```

```
/* Construct the nth binary palindrome with the
   given group number, aux_number and operation
   type */
int constructNthNumber(int group_no, int aux_num,
                                          int op)
{
    int a[INT_SIZE] = {0};

    int num = 0, len_f;
    int i = 0;

     // No need to insert any bit in the middle
    if (op == 2)
    {
        // Length of the final binary representation
        len_f = 2 * group_no;

        // Fill first and last bit as 1
        a[len_f - 1] = a[0] = 1;

        /* Start filling the a[] from middle,
           with the aux_num binary representation */
        while (aux_num)
        {
            // Get the auxiliary number's ith bit and
            // fill around middle
            a[group_no + i] = a[group_no - 1 - i]
                            = aux_num & 1;
            aux_num = aux_num >> 1;
            i++;
        }
    }

    // Insert bit 0 in the middle
    else if (op == 0)
    {
        // Length of the final binary representation
        len_f = 2 * group_no + 1;

        // Fill first and last bit as 1
        a[len_f - 1] = a[0] = 1;
        a[group_no] = 0;

        /* Start filling the a[] from middle, with
        the aux_num binary representation */
        while (aux_num)
        {
            // Get the auxiliary number's ith bit and fill
```

```
            // around middle
            a[group_no + 1 + i] = a[group_no - 1 - i]
                                = aux_num & 1;
            aux_num = aux_num >> 1;
            i++;
        }
    }
    else     // Insert bit 1 in the middle
    {
        // Length of the final binary representation
        len_f = 2 * group_no + 1;

        // Fill first and last bit as 1
        a[len_f - 1] = a[0] = 1;
        a[group_no] = 1;

        /* Start filling the a[] from middle, with
           the aux_num binary representation */
        while (aux_num)
        {
            // Get the auxiliary number's ith bit and fill
            // around middle
            a[group_no + 1 + i] = a[group_no - 1 - i]
                                = aux_num & 1;
            aux_num = aux_num >> 1;
            i++;
        }
    }

    /* Convert the number to decimal from binary */
    for (i = 0; i < len_f; i++)
        num += (1 << i) * a[i];
    return num;
}

/* Will return the nth binary palindrome number */
int getNthNumber(int n)
{
    int group_no = 0, group_offset;
    int count_upto_group = 0, count_temp = 1;
    int op, aux_num;

    /* Add number of elements in all the groups,
       until the group of the nth number is found */
    while (count_temp < n)
    {
        group_no++;
```

```
        // Total number of elements until this group
        count_upto_group = count_temp;
        count_temp += 3 * (1 << (group_no - 1));
    }

    // Element's offset position in the group
    group_offset = n - count_upto_group - 1;

    /* Finding which bit to be placed in the
       middle and finding the number, which we
       will fill from the middle in both
       directions */
    if ((group_offset + 1) <= (1 << (group_no - 1)))
    {
        op = 2; // No need to put extra bit in middle

        // We need to fill this auxiliary number
        // in binary form the middle in both directions
        aux_num = group_offset;
    }
    else
    {
        if (((group_offset + 1) - (1 << (group_no - 1))) % 2)
            op = 0; // Need to Insert 0 at middle
        else
            op = 1; // Need to Insert 1 at middle
        aux_num = ((group_offset) - (1 << (group_no - 1))) / 2;
    }

    return constructNthNumber(group_no, aux_num, op);
}

// Driver code
int main()
{
    int n = 9;
    printf("%d", getNthNumber(n));
    return 0;
}
```

Output :

27

Time complexity of this solution is **O(1)**.

Reference:
https://www.codeproject.com/Articles/1162038/Finding-nth-Binary-Palindrome-in-Csharp

Improved By : vt_m

Source

https://www.geeksforgeeks.org/find-n-th-number-whose-binary-representation-palindrome/

Chapter 167

Find the smallest number with n set and m unset bits

Find the smallest number with n set and m unset bits - GeeksforGeeks

Given two non-negative numbers **n** and **m**. The problem is to find the smallest number having **n** number of set bits and **m** number of unset bits in its binary representation.

Constraints: 1 <= n, 0 <= m, (m+n) <= 31

Note : 0 bits before leading 1 (or leftmost 1) in binary representation are counted

Examples:

```
Input : n = 2, m = 2
Output : 9
(9)10 = (1001)2
We can see that in the binary representation of 9
there are 2 set and 2 unsets bits and it is the
smallest number.

Input : n = 4, m = 1
Output : 23
```

Approach: Following are the steps:

1. Calculate **num** = (1 << (n + m)) – 1. This will produce a number **num** having **(n + m)** number of bits and all are set.
2. Now, toggle bits in the range from **n** to **(n+m-1)** in **num**, i.e, to toggle bits from the rightmost **nth** bit to the rightmost **(n+m-1)th** bit and then return the toggled number. Refer this post.

C/C++

```
 // C++ implementation to find the smallest number
// with n set and m unset bits
#include <bits/stdc++.h>

using namespace std;

// function to toggle bits in the given range
unsigned int toggleBitsFromLToR(unsigned int n,
                                unsigned int l,
                                unsigned int r)
{
    // for invalid range
    if (r < l)
        return n;

    // calculating a number 'num' having 'r'
    // number of bits and bits in the range l
    // to r are the only set bits
    int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

    // toggle bits in the range l to r in 'n'
    // and return the number
    return (n ^ num);
}

// function to find the smallest number
// with n set and m unset bits
unsigned int smallNumWithNSetAndMUnsetBits(unsigned int n,
                                           unsigned int m)
{
    // calculating a number 'num' having '(n+m)' bits
    // and all are set
    unsigned int num = (1 << (n + m)) - 1;

    // required smallest number
    return toggleBitsFromLToR(num, n, n + m - 1);
}

// Driver program to test above
int main()
{
    unsigned int n = 2, m = 2;
    cout << smallNumWithNSetAndMUnsetBits(n, m);
    return 0;
}
```

Java

```
 // Java implementation to find the smallest number
// with n set and m unset bits

class GFG
{
    // Function to toggle bits in the given range
    static int toggleBitsFromLToR(int n, int l, int r)
    {
        // for invalid range
        if (r < l)
            return n;

        // calculating a number 'num' having 'r'
        // number of bits and bits in the range l
        // to r are the only set bits
        int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

        // toggle bits in the range l to r in 'n'
        // and return the number
        return (n ^ num);
    }

    // Function to find the smallest number
    // with n set and m unset bits
    static int smallNumWithNSetAndMUnsetBits(int n, int m)
    {
        // calculating a number 'num' having '(n+m)' bits
        // and all are set
        int num = (1 << (n + m)) - 1;

        // required smallest number
        return toggleBitsFromLToR(num, n, n + m - 1);
    }

    // driver program
    public static void main (String[] args)
    {
        int n = 2, m = 2;
        System.out.println(smallNumWithNSetAndMUnsetBits(n, m));
    }
}

// Contributed by Pramod Kumar
```

Python3

```
 # Python3 implementation to find
# the smallest number with n set
```

```
# and m unset bits

# function to toggle bits in the
# given range
def toggleBitsFromLToR(n, l, r):

    # for invalid range
    if (r < l):
        return n

    # calculating a number 'num'
    # having 'r' number of bits
    # and bits in the range l
    # to r are the only set bits
    num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1)

    # toggle bits in the range
    # l to r in 'n' and return the number
    return (n ^ num)

# function to find the smallest number
# with n set and m unset bits
def smallNumWithNSetAndMUnsetBits(n, m):

    # calculating a number 'num' having
    # '(n+m)' bits and all are set
    num = (1 << (n + m)) - 1

    # required smallest number
    return toggleBitsFromLToR(num, n, n + m - 1);


# Driver program to test above
n = 2
m = 2

ans = smallNumWithNSetAndMUnsetBits(n, m)
print (ans)

# This code is contributed by Saloni Gupta
```

C#

```
 // C# implementation to find the smallest number
// with n set and m unset bits
using System;

class GFG
```

```
{
    // Function to toggle bits in the given range
    static int toggleBitsFromLToR(int n, int l, int r)
    {
        // for invalid range
        if (r < l)
            return n;

        // calculating a number 'num' having 'r'
        // number of bits and bits in the range l
        // to r are the only set bits
        int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

        // toggle bits in the range l to r in 'n'
        // and return the number
        return (n ^ num);
    }

    // Function to find the smallest number
    // with n set and m unset bits
    static int smallNumWithNSetAndMUnsetBits(int n, int m)
    {
        // calculating a number 'num' having '(n+m)' bits
        // and all are set
        int num = (1 << (n + m)) - 1;

        // required smallest number
        return toggleBitsFromLToR(num, n, n + m - 1);
    }

    // Driver program
    public static void Main ()
    {
        int n = 2, m = 2;
        Console.Write(smallNumWithNSetAndMUnsetBits(n, m));
    }
}

// This code is contributed by Sam007
```

Output:

```
9
```

For greater values of **n** and **m**, you can use **long int** and **long long int** datatypes to generate the required number.

Source

https://www.geeksforgeeks.org/find-smallest-number-n-set-m-unset-bits/

Chapter 168

Find the two non-repeating elements in an array of repeating elements

Find the two non-repeating elements in an array of repeating elements - GeeksforGeeks

Asked by SG
Given an array in which all numbers except two are repeated once. (i.e. we have 2n+2 numbers and n numbers are occurring twice and remaining two have occurred once). Find those two numbers in the most efficient way.

Method 1(Use Sorting)
First sort all the elements. In the sorted array, by comparing adjacent elements we can easily get the non-repeating elements. Time complexity of this method is O(nLogn)

Method 2(Use XOR)
Let x and y be the non-repeating elements we are looking for and arr[] be the input array. First calculate the XOR of all the array elements.

```
xor = arr[0]^arr[1]^arr[2].....arr[n-1]
```

All the bits that are set in xor will be set in one non-repeating element (x or y) and not in other. So if we take any set bit of xor and divide the elements of the array in two sets – one set of elements with same bit set and other set with same bit not set. By doing so, we will get x in one set and y in another set. Now if we do XOR of all the elements in first set, we will get first non-repeating element, and by doing same in other set we will get the second non-repeating element.

```
Let us see an example.
   arr[] = {2, 4, 7, 9, 2, 4}
```

```
1) Get the XOR of all the elements.
     xor = 2^4^7^9^2^4 = 14 (1110)
2) Get a number which has only one set bit of the xor.
   Since we can easily get the rightmost set bit, let us use it.
     set_bit_no = xor & ~(xor-1) = (1110) & ~(1101) = 0010
   Now set_bit_no will have only set as rightmost set bit of xor.
3) Now divide the elements in two sets and do xor of
   elements in each set, and we get the non-repeating
   elements 7 and 9. Please see implementation for this
   step.
```

Implementation:

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

/* This finction sets the values of *x and *y to nonr-epeating
 elements in an array arr[] of size n*/
void get2NonRepeatingNos(int arr[], int n, int *x, int *y)
{
  int xor = arr[0]; /* Will hold xor of all elements */
  int set_bit_no;  /* Will have only single set bit of xor */
  int i;
  *x = 0;
  *y = 0;

  /* Get the xor of all elements */
  for(i = 1; i < n; i++)
   xor ^= arr[i];

  /* Get the rightmost set bit in set_bit_no */
  set_bit_no = xor & ~(xor-1);

  /* Now divide elements in two sets by comparing rightmost set
   bit of xor with bit at same position in each element. */
  for(i = 0; i < n; i++)
  {
    if(arr[i] & set_bit_no)
     *x = *x ^ arr[i]; /*XOR of first set */
    else
     *y = *y ^ arr[i]; /*XOR of second set*/
  }
}

/* Driver program to test above function */
int main()
{
  int arr[] = {2, 3, 7, 9, 11, 2, 3, 11};
```

```
  int *x = (int *)malloc(sizeof(int));
  int *y = (int *)malloc(sizeof(int));
  get2NonRepeatingNos(arr, 8, x, y);
  printf("The non-repeating elements are %d and %d", *x, *y);
  getchar();
}
```

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

Please refer below post for detailed explanation :
Find the two numbers with odd occurrences in an unsorted array

Source

https://www.geeksforgeeks.org/find-two-non-repeating-elements-in-an-array-of-repeating-elements/

Chapter 169

Find the winner in nim-game

Find the winner in nim-game - GeeksforGeeks

You are given an array A[] of n-elements. There are two players Alice and Bob. A Player can choose any of element from array and remove it. If the bitwise XOR of all remaining elements equals 0 after removal of selected element, then that player looses. This problem is variation of nim-game.
Note : Each players play game alternately. Find out winner if both of the players play optimally. Alice starts the game first. In case case one-element in array consider its value as the XOR of array.

Examples :

> Input : A[] = {3, 3, 2}
> Output : Winner = Bob
> Explanation : Alice can select 2 and remove it that make XOR of array equals to zero also if Alice choose 3 to remove than Bob can choose any of 2/3 and finally Alice have to make his steps.
>
> Input : A[] = {3, 3}
> Output : Winner = Alice
> Explanation : As XOR of array is already zero Alice can't select any element to remove and hence Alice is winner.

Let's start the solution step by step. We have total of three option for the XOR of array and this game.

1. **XOR of array is already 0:** In this case Alice will unable to make a move and hence Alice is winner.
2. **XOR of array is not zero:** Now, in this case we have two options, either size of array will be odd or even.
   - CASE A: If the array size is odd then for sure Bob will win the game.
   - CASE B: If the array size is even then Alice will win the game.

Above conclusion can be proved with the help of mathematical induction.

Let A[] = {1} i.e. size of array is odd and XOR of array is non-zero: In this case Alice can select element 1 and then A[] will become empty and hence XOR of array can be considered as zero. Resulting Bob as winner.

Let size of array is even and XOR of array is non-zero. Now we can prove that Alice can always find an element to remove such that XOR of remaining elements of array will be non-zero.
To prove this lets start from the contradiction i.e. suppose whatever element you should choose XOR of remaining array must be zero.
So, let A1 Xor A2 Xor … An = X and n is even.
As per our contradiction hypothesis, Ai Xor X = 0 for 1<= i <= n.
Calculate XOR of all X Xor Ai (i.e. n equations),
After taking XOR of all n equations we have X Xor X…Xor X (n-times) = 0 as N is even.
Now, also we have A1 Xor A2 Xor.. An = 0 but we know A1 Xor A2…Xor = X. This means we have at least one element in even-size array such that after its removal XOR of remaining elements in non-zero.

Let size of array is even and XOR of array is non-zero. Alice can not remove an element Ai such that xor of remaining number is zero, because that will make Bob win. Now, take the other case when the xor of remaining N?1 number is non-zero. As we know that N?1 is even and from the induction hypothesis, we can say that the position after the current move will be a winning position for Bob. Hence, it is a losing position for Alice.

```
int res = 0;
for (int i = 1; i <= N; i++) {
    res ^= a[i];
}

if (res == 0)
    return "ALice";
if (N%2 == 0)
    return "Alice";
else
    return "Bob";
```

C++

```
 // CPP to find nim-game winner
#include <bits/stdc++.h>
using namespace std;

// function to find winner of NIM-game
string findWinner(int A[], int n)
{
    int res = 0;
    for (int i = 0; i < n; i++)
```

```
        res ^= A[i];

    // case when Alice is winner
    if (res == 0 || n % 2 == 0)
        return "Alice";

    // when Bob is winner
    else
        return "Bob";
}

// driver program
int main()
{
    int A[] = { 1, 4, 3, 5 };
    int n = siseof(A) / sizeof(A[0]);
    cout << "Winner = " << findWinner(A, n);
    return 0;
}
```

Java

```
 // Java to find nim-game winner
class GFG {

    // function to find winner of NIM-game
    static String findWinner(int A[], int n)
    {
        int res = 0;

        for (int i = 0; i < n; i++)
            res ^= A[i];

        // case when Alice is winner
        if (res == 0 || n % 2 == 0)
            return "Alice";

        // when Bob is winner
        else
            return "Bob";
    }

    //Driver code
    public static void main (String[] args)
    {
        int A[] = { 1, 4, 3, 5 };
        int n =A.length;
```

```
        System.out.print("Winner = "
                    + findWinner(A, n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find nim-game winner

# Function to find winner of NIM-game
def findWinner(A, n):

    res = 0
    for i in range(n):
        res ^= A[i]

    # case when Alice is winner
    if (res == 0 or n % 2 == 0):
        return "Alice"

    # when Bob is winner
    else:
        return "Bob"

# Driver code
A = [ 1, 4, 3, 5 ]
n = len(A)
print("Winner = ", findWinner(A, n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# to find nim-game winner
using System;

class GFG {

    // function to find winner of NIM-game
    static String findWinner(int []A, int n)
    {
        int res = 0;

        for (int i = 0; i < n; i++)
            res ^= A[i];
```

```
        // case when Alice is winner
        if (res == 0 || n % 2 == 0)
            return "Alice";

        // when Bob is winner
        else
            return "Bob";
    }

    //Driver code
    public static void Main ()
    {
        int []A = { 1, 4, 3, 5 };
        int n =A.Length;

        Console.WriteLine("Winner = "
                    + findWinner(A, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP to find nim-game winner

// function to find
// winner of NIM-game
function findWinner($A, $n)
{
    $res = 0;
    for ($i = 0; $i < $n; $i++)
        $res ^= $A[$i];

    // case when Alice is winner
    if ($res == 0 or $n % 2 == 0)
        return "Alice";

    // when Bob is winner
    else
        return "Bob";
}

// Driver Code
$A = array(1, 4, 3, 5 );
$n = count($A);
```

```
echo "Winner = " , findWinner($A, $n);

// This code is contributed by vt_m.
?>
```

Output :

```
Winner = Alice
```

Improved By : vt_m

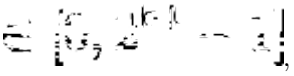
Source

https://www.geeksforgeeks.org/find-winner-nim-game/

Chapter 170

Find two numbers from their sum and XOR

Find two numbers from their sum and XOR - GeeksforGeeks

Given the sum and xor of two numbers **X** and **Y** s.t. sum and xor  , we need to find the numbers minimizing the value of **X**.

Examples :

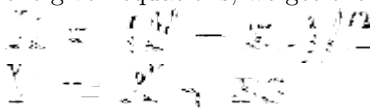
```
Input : sum = 17
        xor = 13
Output : X = 2
         Y = 15

Input : sum = 1870807699
        xor = 259801747
Output : X = 805502976
         Y = 1065304723

Input : sum = 1639
        xor = 1176
Output : No such numbers exist
```

Let the summation be S and xor be xo. The summation and xor operations both are commutative in nature. This means that for a corresponding set bit in X if we swap it with an unset bit at same position in Y, the sum and xor would remain unaffected. The only affect it would make is that it will reduce the value of X and increase the value of Y.

For example, X = 7 and Y = 10, X+Y = S = 17 and X^Y = xo = 13.
Binary representation of X = 0111

Binary representation of Y = 1010
Now, for a given position in X which is 1, if for the same position in Y is 0 and we swap it;
New binary representation of X = 0010
New binary representation of Y = 1111
New value of X = 2
New Value of Y = 15
Their sum and xor are still the same.

Now, let's use the above piece of information to exploit our problem statement because following the above steps, we have successfully minimized the value of X. We can even conclude from the above proof that $\mathbf{Y = xo + X}$
Now, we know that $\mathbf{X + Y = S}$ and we also know that $\mathbf{Y = X + xo}$, now if we equate the given equations, we get the results are



Using the above equations we can easily compute the required values of X and Y.

C++

```
 // CPP program to find two numbers with
// given Sum and XOR such that value of
// first number is minimum.
#include <iostream>
using namespace std;

// Function that takes in the sum and XOR
// of two numbers and generates the two
// numbers such that the value of X is
// minimized
void compute(unsigned long int S,
             unsigned long int Xo)
{
    // If sum becomes less than Xor
    // in this case, no solution shall
    // exist.
    if (S < Xo)
        cout << "No such numbers exist";
    else {

        // Application of the derived formula!
        int x = (S - Xo) / 2;
        cout << "X = " << x << endl;
        cout << "Y = " << (x + Xo) << endl;
    }
}

// Driver function
int main()
```

```
{
    unsigned long int S = 17, Xo = 13;
    compute(S, Xo);
    return 0;
}
```

Java

```
 // Java program to find two numbers
// with given Sum and XOR such that
// value of first number is minimum.
class GFG{

    // Function that takes in the
    // sum and XOR of two numbers
    // and generates the two numbers
    // such that the value of X is
    // minimized
    static void compute(int S, int Xo)
    {

        // If sum becomes less than
        // Xor in this case, no solution
        // shall exist.
        if (S < Xo)
            System.out.printf
              ("No such numbers exist");
        else {

            // Application of the derived
            // formula!
            int x = (S - Xo) / 2;
            System.out.printf
                        ( "X = %d\n", x);
            System.out.printf
                    ( "Y = %d", (x + Xo));
        }
    }

    // Driver function
    public static void main(String[] args)
    {
        int S = 17, Xo = 13;

        compute(S, Xo);
    }
}
```

```
// This code is contributed by
// Smitha Dinesh Semwal.
```

Python3

```
 # Python program to find
# two numbers with
# given Sum and XOR such
# that value of
# first number is minimum.

# Function that takes in
# the sum and XOR
# of two numbers and
# generates the two
# numbers such that the
# value of X is
# minimized
def compute(S, Xo):

    # If sum becomes less than Xor
    # in this case, no solution shall
    # exist.
    if (S < Xo):
        print("No such numbers exist")
    else :

        # Application of the derived formula!
        x = int((S - Xo) / 2)
        print("X =",x)
        print("Y =",(x + Xo))


# Driver function
S = 17
Xo = 13
compute(S, Xo)

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to find two numbers
// with given Sum and XOR such that
// value of first number is minimum.
using System;
```

```
class GFG
{

    // Function that takes in the
    // sum and XOR of two numbers
    // and generates the two numbers
    // such that the value of X is
    // minimized
    public static void compute(int S,
                               int Xo)
    {

        // If sum becomes less than
        // Xor in this case, no
        // solution shall exist.
        if (S < Xo)
        Console.WriteLine("No such " +
                     "numbers exist");
        else
        {

        // Application of the
        // derived formula!
        int x = (S - Xo) / 2;
        Console.WriteLine("X = "+ x);
        Console.WriteLine("Y = "+ (x + Xo));
        }
    }

    // Driver Code
    static public void Main ()
    {
        int S = 17, Xo = 13;

        compute(S, Xo);
    }
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP program to find two numbers
// with given Sum and XOR such that
// value of first number is minimum.
```

```
// Function that takes in the sum
// and XOR of two numbers and
// generates the two numbers such
// that the value of X is minimized
function compute($S, $Xo)
{
    // If sum becomes less than
    // Xor in this case, no
    // solution shall exist.
    if ($S < $Xo)
        echo "No such numbers exist";
    else
    {

        // Application of the
        // derived formula!
        $x = ($S - $Xo) / 2;
        echo "X = " , $x , "\n";
        echo "Y = " , ($x + $Xo) , "\n";
    }
}

// Driver Code
$S = 17;
$Xo = 13;
compute($S, $Xo);

// This code is contributed by ajit
?>
```

Output :

```
X = 2
Y = 15
```

Time complexity of the above approach $O(1)$.

Improved By : jit_t, Rajnish Ranjan

Source

https://www.geeksforgeeks.org/find-two-numbers-sum-xor/

Chapter 171

Find value of k-th bit in binary representation

Find value of k-th bit in binary representation - GeeksforGeeks

Given a number n and k (1 <= k <= 32), find value of k-th bit in binary representation of n. Bits are numbered from right (Least Significant Bit) to left (Most Significant Bit).

Examples :

```
Input :  n = 13, k = 2
Output : 0
Binary representation of 13 is 1101.
Second bit from right is 0.

Input :  n = 14, k = 3
Output : 1
Binary representation of 14 is 1110.
Third bit from right is 1.
```

1) Find a number with all 0s except k-th position. We get this number using (1 << (k-1)). For example if k = 3, then (1 << 2) gives us (00..00100).
2) Do bitwise and of above obtained number with n to find if k-th bit in n is set or not.

C++

```
 // CPP program to find k-th bit from right
#include <bits/stdc++.h>
using namespace std;

void printKthBit(unsigned int n, unsigned int k)
```

```
{
     cout << (n & (1 << (k-1)));
}

int main()
{
  unsigned int n = 13, k = 2;
  printKthBit(n, k);
  return 0;
}
```

Java

```
 // Java program to find
// k-th bit from right
import java.io.*;

class GFG
{
    static void printKthBit(long n, long k)
    {
    System.out.println( (n & (1 << (k - 1))));
    }


// Driver Code
public static void main (String[] args)
{
    long n = 13, k = 2;
    printKthBit(n, k);
}
}

// This code is contributed by anuj_67.
```

Python3

```
 # Python 3 program to find
# k-th bit from right
def printKthBit(n, k):

    print(n & (1 << (k-1)))

n = 13
k = 2
printKthBit(n, k)
```

```
# This code is contributed
# by Smitha
```

C#

```
 // C# program to find k-th bit from right
using System;

class GFG {

    static void printKthBit(long n, long k)
    {
        Console.WriteLine( (n & (1 <<
                            (int)(k - 1))));
    }

    // Driver Code
    public static void Main ()
    {
        long n = 13, k = 2;

        printKthBit(n, k);
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to find
// k-th bit from right

function printKthBit($n, $k)
{
    echo ($n & (1 << ($k - 1)));
}

// Driver Code
$n = 13; $k = 2;
printKthBit($n, $k);

// This code is contributed
// by anuj_67.
?>
```

Output :

0

Improved By : vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/find-value-k-th-bit-binary-representation/

Chapter 172

Find whether a given number is a power of 4 or not

Find whether a given number is a power of 4 or not - GeeksforGeeks

Given an integer n, find whether it is a power of 4 or not.

Example :

```
Input : 16
Output : 16 is a power of 4

Input : 20
Output : 20 is not a power of 4
```

1. A simple method is to take log of the given number on base 4, and if we get an integer then number is power of 4.

2. Another solution is to keep dividing the number by 4, i.e, do n = n/4 iteratively. In any iteration, if n%4 becomes non-zero and n is not 1 then n is not a power of 4, otherwise n is a power of 4.

C

```
 #include<stdio.h>
#define bool int

/* Function to check if x is power of 4*/
bool isPowerOfFour(int n)
{
  if(n == 0)
    return 0;
```

```
  while(n != 1)
  {
   if(n % 4 != 0)
      return 0;
    n = n / 4;
  }
  return 1;
}

/*Driver program to test above function*/
int main()
{
  int test_no = 64;
  if(isPowerOfFour(test_no))
    printf("%d is a power of 4", test_no);
  else
    printf("%d is not a power of 4", test_no);
  getchar();
}
```

Java

```
 // Java code to check if given
// number is power of 4 or not

class GFG {

    // Function to check if
    // x is power of 4
    static int isPowerOfFour(int n)
    {
        if(n == 0)
        return 0;
        while(n != 1)
        {
            if(n % 4 != 0)
            return 0;
            n = n / 4;
        }
        return 1;
    }

    // Driver program
    public static void main(String[] args)
    {
        int test_no = 64;
        if(isPowerOfFour(test_no) == 1)
         System.out.println(test_no +
```

```
                            " is a power of 4");
        else
         System.out.println(test_no +
                            "is not a power of 4");
    }
}

// This code is contributed
// by  prerna saini
```

Python3

```
 # Python3 program to check if given
# number is power of 4 or not

# Function to check if x is power of 4
def isPowerOfFour(n):
    if (n == 0):
        return False
    while (n != 1):
            if (n % 4 != 0):
                return False
            n = n // 4

    return True

# Driver code
test_no = 64
if(isPowerOfFour(64)):
    print(test_no, 'is a power of 4')
else:
    print(test_no, 'is not a power of 4')

# This code is contributed by Danish Raza
```

C#

```
 // C# code to check if given
// number is power of 4 or not
using System;

class GFG {

    // Function to check if
    // x is power of 4
    static int isPowerOfFour(int n)
    {
```

```
        if (n == 0)
            return 0;
        while (n != 1) {
            if (n % 4 != 0)
                return 0;
            n = n / 4;
        }
        return 1;
    }

    // Driver code
    public static void Main()
    {
        int test_no = 64;
        if (isPowerOfFour(test_no) == 1)
            Console.Write(test_no +
                    " is a power of 4");
        else
            Console.Write(test_no +
                " is not a power of 4");
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP code to check if given
// number is power of 4 or not

// Function to check if
// x is power of 4
function isPowerOfFour($n)
{

    if($n == 0)
        return 0;
    while($n != 1)
    {
        if($n % 4 != 0)
            return 0;
            $n = $n / 4;
    }
    return 1;
}

// Driver Code
```

```
$test_no = 64;

if(isPowerOfFour($test_no))
    echo $test_no," is a power of 4";
else
    echo $test_no," is not a power of 4";

// This code is contributed by Rajesh
?>
```

Output :

```
64 is a power of 4
```

3. A number n is a power of 4 if following conditions are met.
a) There is only one bit set in the binary representation of n (or n is a power of 2)
b) The count of zero bits before the (only) set bit is even.

For example: 16 (10000) is power of 4 because there is only one bit set and count of 0s before the set bit is 4 which is even.

Thanks to Geek4u for suggesting the approach and providing the code.

C

```
 #include<stdio.h>
#define bool int

bool isPowerOfFour(unsigned int n)
{
  int count = 0;

  /*Check if there is only one bit set in n*/
  if ( n && !(n&(n-1)) )
  {
     /* count 0 bits before set bit */
     while(n > 1)
     {
       n  >>= 1;
       count += 1;
     }

    /*If count is even then return true else false*/
    return (count%2 == 0)? 1 :0;
  }
```

```
  /* If there are more than 1 bit set
    then n is not a power of 4*/
  return 0;
}

/*Driver program to test above function*/
int main()
{
   int test_no = 64;
   if(isPowerOfFour(test_no))
     printf("%d is a power of 4", test_no);
   else
     printf("%d is not a power of 4", test_no);
   getchar();
}
```

Java

```
 // Java program to check
// if given number is
// power of 4 or not
import java.io.*;
class GFG
{
    static int isPowerOfFour(int n)
    {
        int count = 0;

        /*Check if there is
        only one bit set in n*/
        int x = n & (n - 1);

        if ( n > 0 && x == 0)
        {
            /* count 0 bits
            before set bit */
            while(n > 1)
            {
                n >>= 1;
                count += 1;
            }

            /*If count is even
            then return true
            else false*/
            return (count % 2 == 0) ? 1 : 0;
        }
```

```
            /* If there are more than
            1 bit set then n is not a
            power of 4*/
        return 0;
    }

    // Driver Code
    public static void main(String[] args)
    {
        int test_no = 64;

        if(isPowerOfFour(test_no)>0)
            System.out.println(test_no +
                               " is a power of 4");
        else
            System.out.println(test_no +
                               " is not a power of 4");
    }
}

// This code is contributed by mits
```

Python3

```
 # Python3 program to check if given
# number is power of 4 or not

# Function to check if x is power of 4
def isPowerOfFour(n):

    count = 0

    # Check if there is only one
    # bit set in n
    if (n and (not(n & (n - 1)))):

        # count 0 bits before set bit
        while(n > 1):
            n >>= 1
            count += 1

        # If count is even then return
        # true else false
        if(count % 2 == 0):
            return True
        else:
            return False
```

```
# Driver code
test_no = 64
if(isPowerOfFour(64)):
    print(test_no, 'is a power of 4')
else:
    print(test_no, 'is not a power of 4')

# This code is contribued by Danish Raza
```

C#

```
 // C# program to check if given
// number is power of 4 or not
using System;

class GFG {

    static int isPowerOfFour(int n)
    {
        int count = 0;

        /*Check if there is only one bit
        set in n*/
        int x = n & (n-1);

        if ( n > 0 && x == 0)
        {
            /* count 0 bits before set bit */
            while(n > 1)
            {
                n >>= 1;
                count += 1;
            }

            /*If count is even then return
            true else false*/
            return (count % 2 == 0) ? 1 : 0;
        }

            /* If there are more than 1 bit set
            then n is not a power of 4*/
        return 0;
    }

    /*Driver program to test above function*/
    static void Main()
    {
        int test_no = 64;
```

```
        if(isPowerOfFour(test_no)>0)
            Console.WriteLine("{0} is a power of 4",
                                        test_no);
        else
            Console.WriteLine("{0} is not a power of 4",
                                          test_no);
    }
}

// This Code is Contributed by mits
```

PHP

```
 <?php

function isPowerOfFour($n)
{
    $count = 0;

/*Check if there is only one bit set in n*/
if ( $n && !($n&($n-1)) )
{
    /* count 0 bits before set bit */
    while($n > 1)
    {
    $n >>= 1;
    $count += 1;
    }

    /*If count is even then return true else false*/
    return ($count%2 == 0)? 1 :0;
}

/* If there are more than 1 bit set
    then n is not a power of 4*/
return 0;
}

/*Driver program to test above function*/

    $test_no = 64;

if(isPowerOfFour($test_no))

    echo $test_no, " is a power of 4";

else
```

```
    echo $test_no, " not is a power of 4";


#This Code is Contributed by Ajit
?>
```

Output:

```
64 is a power of 4
```

Improved By : kisanrajesh, jit_t, Mithun Kumar

Source

https://www.geeksforgeeks.org/find-whether-a-given-number-is-a-power-of-4-or-not/

Chapter 173

Finding the Parity of a number Efficiently

Finding the Parity of a number Efficiently - GeeksforGeeks

Given an integer N. The task is to write a program to find the parity of the given number.

Note: Parity of a number is used to define if the total number of set-bits(1-bit in binary representation) in a number is even or odd. If the total number of set-bits in the binary representation of a number is even then the number is said to have even parity, otherwise, it will have odd parity.

Examples:

```
Input : N = 13
Output : Odd Parity
Bianry representation of 13 is (1101)

Input : N = 9 (1001)
Output : Even Parity
```

The parity of a number represented by 32-bits can be efficiently calculated by performing the following operations.

Let the given number be x, then perform the below operations:

- y = x^(x>>1)
- y = y^(y>>2)
- y = y^(y>>4)
- y = y^(y>>8)
- y = y^(y>>16)

Now, the rightmost bit in y will represent the parity of x. If the rightmost bit is 1, then x will have odd parity and if it is 0 then x will have even parity.

So, in order to extract the last bit of y, perform bit-wise AND operation of y with 1.

```
if(y&1==1)
    odd Parity
else
    even Parity
```

Below is the implementation of the above approach:

C++

```
 // Program to find the parity of a given number
#include <bits/stdc++.h>

using namespace std;

// Function to find the parity
bool findParity(int x)
{
    int y = x ^ (x >> 1);
    y = y ^ (y >> 2);
    y = y ^ (y >> 4);
    y = y ^ (y >> 8);
    y = y ^ (y >> 16);

    // Rightmost bit of y holds the parity value
    // if (y&1) is 1 then parity is odd else even
    if (y & 1)
        return 1;
    return 0;
}

// Driver code
int main()
{
    (findParity(9)==0)?cout<<"Even Parity\n":
                            cout<<"Odd Parity\n";

    (findParity(13)==0)?cout<<"Even Parity\n":
                            cout<<"Odd Parity\n";

    return 0;
}
```

Java

```
 // Program to find the
// parity of a given number
import java.io.*;

class GFG
{

// Function to find the parity
static boolean findParity(int x)
{
    int y = x ^ (x >> 1);
        y = y ^ (y >> 2);
        y = y ^ (y >> 4);
        y = y ^ (y >> 8);
        y = y ^ (y >> 16);

    // Rightmost bit of y holds
    // the parity value
    // if (y&1) is 1 then parity
    // is odd else even
    if ((y & 1) > 0)
        return true;
    return false;
}

// Driver code
public static void main (String[] args)
{
    if((findParity(9) == false))
        System.out.println("Even Parity");
    else
        System.out.println("Odd Parity");

    if(findParity(13) == false)
        System.out.println("Even Parity");
    else
        System.out.println("Odd Parity");
}
}

// This Code is Contributed by chandan_jnu.
```

Python3

```
# Program to find the
# parity of a given number
```

```
# Function to find the parity
def findParity(x):
y = x ^ (x >> 1);
y = y ^ (y >> 2);
y = y ^ (y >> 4);
y = y ^ (y >> 8);
y = y ^ (y >> 16);

# Rightmost bit of y holds
# the parity value if (y&1)
# is 1 then parity is odd
# else even
if (y & 1):
return 1;
return 0;

# Driver code
if(findParity(9) == 0):
print("Even Parity");
else:
print("Odd Parity\n");

if(findParity(13) == 0):
print("Even Parity");
else:
print("Odd Parity");

# This code is contributed by mits
```

C#

```
 // Program to find the
// parity of a given number
using System;

class GFG
{

// Function to find the parity
static bool findParity(int x)
{
    int y = x ^ (x >> 1);
        y = y ^ (y >> 2);
        y = y ^ (y >> 4);
        y = y ^ (y >> 8);
        y = y ^ (y >> 16);

    // Rightmost bit of y holds
    // the parity value
    // if (y&1) is 1 then parity
```

```
    // is odd else even
    if ((y & 1) > 0)
        return true;
    return false;
}

// Driver code
public static void Main ()
{
    if((findParity(9) == false))
        Console.WriteLine("Even Parity");
    else
        Console.WriteLine("Odd Parity");

    if(findParity(13) == false)
        Console.WriteLine("Even Parity");
    else
        Console.WriteLine("Odd Parity");
}
}

// This Code is Contributed
// by chandan_jnu
```

PHP

```
 <?php
// Program to find the
// parity of a given number

// Function to find the parity
function findParity($x)
{
    $y = $x ^ ($x >> 1);
    $y = $y ^ ($y >> 2);
    $y = $y ^ ($y >> 4);
    $y = $y ^ ($y >> 8);
    $y = $y ^ ($y >> 16);

    // Rightmost bit of y holds
    // the parity value if (y&1)
    // is 1 then parity is odd
    // else even
    if ($y & 1)
        return 1;
    return 0;
}
```

```
// Driver code
(findParity(9) == 0) ?
 print("Even Parity\n"):
 print("Odd Parity\n");

(findParity(13) == 0) ?
print("Even Parity\n"):
print("Odd Parity\n");

// This Code is Contributed by mits
?>
```

Output:

```
Even Parity
Odd Parity
```

Improved By : Mithun Kumar, Chandan_Kumar

Source

https://www.geeksforgeeks.org/finding-the-parity-of-a-number-efficiently/

Chapter 174

First element greater than or equal to X in prefix sum of N numbers using Binary Lifting

First element greater than or equal to X in prefix sum of N numbers using Binary Lifting - GeeksforGeeks

Given an array of N integers and a number X. The task is to find the index of first element which is greater than or equal to X in prefix sums of N numbers.

Examples:

> **Input:** arr[] = { 2, 5, 7, 1, 6, 9, 12, 4, 6 } and x = 8
> **Output:** 2
> prefix sum array formed is { 2, 7, 14, 15, 21, 30, 42, 46, 52}, hence 14 is the number whose index is 2
>
> **Input:** arr[] = { 2, 5, 7, 1, 6, 9, 12, 4, 6 } and x = 30
> **Output:** 5

Approach: The problem can be solved using lower_bound function in Binary search. But in this post, the problem will be solved using **Binary-Lifting**. In binary lifting, a value is increased (or lifted) by powers of 2, starting with the highest possible power of *2(log(N)) down to the lowest power(0).*

- initialize position = 0 and set each bit of position, from most significant bit to least significant bit.
- whenever a bit is set to 1, the value of position increases (or lifts).
- while increasing or lifting position, make sure that prefix sum till position should be less than v.
- here, log(N) bits are needed for all possible values of 'position' **(from log(N)th to 0th bit).**

- determine the value of the i-th bit. First, check if setting the i-th bit won't make 'position' greater than N, which is the size of the array. Before lifting to the new 'position', check that value at that new 'position' is less than X or not.
- if this condition is true, then target position lies above the **'position' + 2^i**, but below the 'position' + 2^(i+1). This is because if target position was above **'position' + 2^(i+1),** then the position would have been already lifted by **2^(i+1)** (this logic is similar to binary lifting in trees).
- if it is false, then target value lies between **'position' and 'position' + 2^i**, so try to lift by a lower power of 2. Finally loop will end such that value at that position is less than X. Here, in this question, the lower bound is asked. so, return **'position' + 1**.

Below is the implementation of the above approach:

```
 // CPP program to find lower_bound of x in
// prefix sums array using binary lifting.
#include <bits/stdc++.h>
using namespace std;

// function to make prefix sums array
void MakePreSum(int arr[], int presum[], int n)
{
    presum[0] = arr[0];
    for (int i = 1; i < n; i++)
        presum[i] = presum[i - 1] + arr[i];
}

// function to find lower_bound of x in
// prefix sums array using binary lifting.
int BinaryLifting(int presum[], int n, int x)
{
    // intisalize position
    int pos = 0;

    // find log to the base 2 value of n.
    int LOGN = log2(n);

    // if x less than first number.
    if (x <= presum[0])
        return 0;

    // starting from most significant bit.
    for (int i = LOGN; i >= 0; i--) {

        // if value at this position less
        // than x then updateposition
        // Here (1<<i) is similar to 2^i.
        if (pos + (1 << i) < n &&
```

```
            presum[pos + (1 << i)] < x) {
            pos += (1 << i);
        }
    }

    // +1 because 'pos' will have position
    // of largest value less than 'x'
    return pos + 1;
}

// Driver code
int main()
{
    // given array
    int arr[] = { 2, 5, 7, 1, 6, 9, 12, 4, 6 };

    // value to find
    int x = 8;

    // size of array
    int n = sizeof(arr) / sizeof(arr[0]);

    // to store prefix sum
    int presum[n] = { 0 };

    // call for prefix sum
    MakePreSum(arr, presum, n);

    // fucntion call
    cout << BinaryLifting(presum, n, x);

    return 0;
}
```

Output:

```
2
```

Time Complexity: O(N)
Auxiliary Space: O(N)

Source

https://www.geeksforgeeks.org/first-element-greater-than-or-equal-to-x-in-prefix-sum-of-n-numbers-using-binary-l

Chapter 175

For every set bit of a number toggle bits of other

For every set bit of a number toggle bits of other - GeeksforGeeks

Given two integer numbers, whenever the bits of the first number is set i.e. 1, toggle the bits of the second number leaving the rest bits of the second number unchanged.
Examples :

```
Input: 2 5
Output: 7
2 is represented as 10 in binary and 5
is represented as 101. Hence toggling the
2nd bit of 5 from right, thus the new
number becomes 7 i.e. 111

Input: 1 3
Output: 2
```

Approach:
Just do XOR of the given two Number.

CPP

```
 // A CPP program toggle bits of n2
// that are at same position as set
// bits of n1.
#include <bits/stdc++.h>
using namespace std;
```

```
// function for the Nega_bit
int toggleBits(int n1, int n2)
{
    return n1 ^ n2;
}

// Driver program to test above
int main()
{
    int n1 = 2, n2 = 5;
    cout << toggleBitst(n1, n2) << endl;
    return 0;
}
```

Java

```
 // A Java program toggle bits of n2
// that are at same position as set
// bits of n1.
import java.io.*;

class GFG {

    // function for the Nega_bit
    static int toggleBits(int n1, int n2)
    {
        return (n1 ^ n2);
    }

    // Driver program
    public static void main(String args[])
    {
        int n1 = 2, n2 = 5;
        System.out.println(toggleBits(n1, n2));
    }
}




// This code is contributed
// by Nikita Tiwari.
```

Python3

```
 # A Python 3 program toggle bits of n2
# that are at same position as set
```

```
# bits of n1.


# function for the Nega_bit
def toggleBits(n1, n2) :
    return (n1 ^ n2)


# Driver program to test above
n1 = 2
n2 = 5

print(toggleBits(n1, n2))

# This code is contributed
# by Nikita Tiwari.
```

C#

```
 // C# program toggle bits of n2
// that are at same position as set
// bits of n1.
using System;

class GFG {

    // function for the Nega_bit
    static int toggleBits(int n1, int n2)
    {
        return (n1 ^ n2);
    }

    // Driver program
    public static void Main()
    {

        int n1 = 2, n2 = 5;

        Console.WriteLine(toggleBits(n1, n2));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
```

```
// PHP program to toggle bits of n2
// that are at same position as set
// bits of n1.

// function for the Nega_bit
function toggleBits($n1, $n2)
{
    return $n1 ^ $n2;
}

// Driver code
$n1 = 2;
$n2 = 5;
echo toggleBits($n1, $n2)."\n";

// This code is contributed by mits
?>
```

Output :

```
7
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/for-every-set-bit-of-a-number-toggle-bits-of-other/

Chapter 176

Game of Nim with removal of one stone allowed

Game of Nim with removal of one stone allowed - GeeksforGeeks

In Game of Nim, two players take turns removing objects from heaps or the pile of stones. Suppose two players A and B are playing the game. Each is allowed to take only one stone from the pile. The player who picks the last stone of the pile will win the game. Given **N** the number of stones in the pile, the task is to find the winner, if player A starts the game.

Examples :

```
Input : N = 3.
Output : Player A

Player A remove stone 1 which is at the top, then Player B remove stone 2
and finally player A removes the last stone.

Input : N = 15.
Output : Player A
```

For N = 1, player A will remove the only stone from the pile and wins the game.
For N = 2, player A will remove the first stone and then player B remove the second or the last stone. So player B will win the game.

So, we can observe player A wins when N is odd and player B wins when N is even.

Below is the implementation of this approach:

C++

```
// C++ program for Game of Nim with removal
```

```
// of one stone allowed.
#include<bits/stdc++.h>
using namespace std;

// Return true if player A wins,
// return false if player B wins.
bool findWinner(int N)
{
  // Checking the last bit of N.
  return N&1;
}

// Driven Program
int main()
{
  int N = 15;
  findWinner(N)? (cout << "Player A";):
                 (cout << "Player B";);
  return 0;
}
```

Java

```
 // JAVA Code For Game of Nim with
// removal of one stone allowed
import java.util.*;

class GFG {

    // Return true if player A wins,
    // return false if player B wins.
    static int findWinner(int N)
    {
      // Checking the last bit of N.
      return N & 1;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int N = 15;
        if(findWinner(N)==1)
            System.out.println("Player A");
        else
             System.out.println("Player B");

    }
}
```

```
// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python3 code for Game of Nim with
# removal of one stone allowed.

# Return true if player A wins,
# return false if player B wins.
def findWinner( N ):

    # Checking the last bit of N.
    return N & 1

# Driven Program
N = 15
print("Player A" if findWinner(N) else "Player B")

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# Code For Game of Nim with
// removal of one stone allowed
using System;

class GFG {

    // Return true if player A wins,
    // return false if player B wins.
    static int findWinner(int N)
    {
        // Checking the last bit of N.
        return N & 1;
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int N = 15;

        if(findWinner(N) == 1)
            Console.Write("Player A");
        else
            Console.Write("Player B");
```

```
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program for Game of
// Nim with removal of one
// stone allowed.

// Return true if player A wins,
// return false if player B wins.
function findWinner($N)
{

// Checking the last bit of N.
return $N&1;
}

// Driver Code
$N = 15;

if(findWinner($N))
echo "Player A";
else
echo "Player B";

// This code is contributed by vt_m.
?>
```

Output :

```
Player A
```

Time Complexity: O(1).

Improved By : vt_m

Source

https://www.geeksforgeeks.org/game-nim-removal-one-stone-allowed/

Chapter 177

Generate 0 and 1 with 25% and 75% probability

Generate 0 and 1 with 25% and 75% probability - GeeksforGeeks

Given a function rand50() that returns 0 or 1 with equal probability, write a function that returns 1 with 75% probability and 0 with 25% probability using rand50() only. Minimize the number of calls to rand50() method. Also, use of any other library function and floating point arithmetic are not allowed.

The idea is to use **Bitwise OR**. A bitwise OR takes two bits and returns 0 if both bits are 0, while otherwise the result is 1. So it has 75% probability that it will return 1.

Below is the implementation of above idea :

C++

```
 // Program to print 1 with 75% probability and 0
// with 25% probability
#include <iostream>
using namespace std;

// Random Function to that returns 0 or 1 with
// equal probability
int rand50()
{
    // rand() function will generate odd or even
    // number with equal probability. If rand()
    // generates odd number, the function will
    // return 1 else it will return 0.
    return rand() & 1;
}

// Random Function to that returns 1 with 75%
```

```
// probability and 0 with 25% probability using
// Bitwise OR
bool rand75()
{
    return rand50() | rand50();
}

// Driver code to test above functions
int main()
{
    // Intialize random number generator
    srand(time(NULL));

    for(int i = 0; i < 50; i++)
        cout << rand75();

    return 0;
}
```

PHP

```
 <?php
// Program to print 1 with 75% probability
// and 0 with 25% probability

// Random Function to that returns 0 or
// 1 with equal probability
function rand50()
{

    // rand() function will generate
    // odd or even number with equal
    // probability. If rand() generates
    // odd number, the function will
    // return 1 else it will return 0.
    return rand() & 1;
}

// Random Function to that returns
// 1 with 75% probability and 0
// with 25% probability using
// Bitwise OR
function rand75()
{
    return rand50() | rand50();
}

    // Driver Code
```

```
    // Intialize random
    // number generator
    srand(time(NULL));

    for($i = 0; $i < 50; $i++)
        echo rand75();

// This code is contributed m_kit
?>
```

Output:

```
11101111110010010110011111111101111110111100011000
```

On similar lines, we can also use **Bitwise AND**. Since it returns 0 with 75% probability, we have to invert the result.

```
// Random Function to that returns 1 with 75%
// probability and 0 with 25% probability using
// Bitwise AND
bool rand75()
{
    return !(rand50() & rand50());
}
```

We can replace Bitwise OR and Bitwise AND operator by **OR and AND operators** as well –

```
// Random Function to that returns 1 with 75%
// probability and 0 with 25% probability using
// OR or AND operator
int rand75()
{
    return !(rand50() && rand50());
    // return rand50() || rand50()
}
```

We can also achieve the result using **left shift operator and Bitwise XOR** –

C++

```
 // Program to print 1 with 75% probability and 0
// with 25% probability
```

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

// Random Function to that returns 0 or 1 with
// equal probability
int rand50()
{
    // rand() function will generate odd or even
    // number with equal probability. If rand()
    // generates odd number, the function will
    // return 1 else it will return 0.
    return rand() & 1;
}

// Random Function to that returns 1 with 75%
// probability and 0 with 25% probability using
// left shift and Bitwise XOR
int rand75()
{
    // x is one of {0, 1}
    int x = rand50();

    x = x << 1;

    // x is now one of {00, 10}

    x = x ^ rand50();

    // x is now one of {00, 01, 10, 11}

    return (x > 0) ? 1 : 0;
}

// Driver code to test above functions
int main()
{
    // Intialize random number generator
    srand(time(NULL));

    for (int i = 0; i < 50; i++)
        cout << rand75();

    return 0;
}
```

PHP

```
 <?php
```

```
// Program to print 1 with
// 75% probability and 0
// with 25% probability

// Random Function to that
// returns 0 or 1 with
// equal probability
function rand50()
{
    // rand() function will
    // generate odd or even
    // number with equal
    // probability. If rand()
    // generates odd number,
    // the function will return
    // 1 else it will return 0.
    return rand() & 1;
}

// Random Function to that
// returns 1 with 75%
// probability and 0 with
// 25% probability using
// left shift and Bitwise XOR
function rand75()
{
    // x is one of {0, 1}
    $x = rand50();

    $x = $x << 1;

    // x is now one
    // of {00, 10}

    $x = $x ^ rand50();

    // x is now one of
    // {00, 01, 10, 11}

    return ($x > 0) ? 1 : 0;
}

// Driver code

// Intialize random
// number generator
srand(time(NULL));
```

```
for ($i = 0; $i < 50; $i++)
    echo rand75();

// This code is contributed
// by ajit
?>
```

Output:

```
01101110111011000111111111110001111011101110110110
```

Please note above solutions will produce **different results** every time we run them.

Improved By : jit_t

Source

https://www.geeksforgeeks.org/generate-0-1-25-75-probability/

Chapter 178

Generate n-bit Gray Codes

Generate n-bit Gray Codes - GeeksforGeeks

Given a number n, generate bit patterns from 0 to 2^n-1 such that successive patterns differ by one bit.

Examples:

```
Following is 2-bit sequence (n = 2)
  00 01 11 10
Following is 3-bit sequence (n = 3)
  000 001 011 010 110 111 101 100
And Following is 4-bit sequence (n = 4)
  0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111
  1110 1010 1011 1001 1000
```

The above sequences are Gray Codes of different widths. Following is an interesting pattern in Gray Codes.

n-bit Gray Codes can be generated from list of (n-1)-bit Gray codes using following steps.
1) Let the list of (n-1)-bit Gray codes be L1. Create another list L2 which is reverse of L1.
2) Modify the list L1 by prefixing a '0' in all codes of L1.
3) Modify the list L2 by prefixing a '1' in all codes of L2.
4) Concatenate L1 and L2. The concatenated list is required list of n-bit Gray codes.

For example, following are steps for generating the 3-bit Gray code list from the list of 2-bit Gray code list.
L1 = {00, 01, 11, 10} (List of 2-bit Gray Codes)
L2 = {10, 11, 01, 00} (Reverse of L1)
Prefix all entries of L1 with '0', L1 becomes {000, 001, 011, 010}
Prefix all entries of L2 with '1', L2 becomes {110, 111, 101, 100}
Concatenate L1 and L2, we get {000, 001, 011, 010, 110, 111, 101, 100}

To generate n-bit Gray codes, we start from list of 1 bit Gray codes. The list of 1 bit Gray code is {0, 1}. We repeat above steps to generate 2 bit Gray codes from 1 bit Gray codes, then 3-bit Gray codes from 2-bit Gray codes till the number of bits becomes equal to n. Following is C++ implementation of this approach.

```
 // C++ program to generate n-bit Gray codes
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// This function generates all n bit Gray codes and prints the
// generated codes
void generateGrayarr(int n)
{
    // base case
    if (n <= 0)
        return;

    // 'arr' will store all generated codes
    vector<string> arr;

    // start with one-bit pattern
    arr.push_back("0");
    arr.push_back("1");

    // Every iteration of this loop generates 2*i codes from previously
    // generated i codes.
    int i, j;
    for (i = 2; i < (1<<n); i = i<<1)
    {
        // Enter the prviously generated codes again in arr[] in reverse
        // order. Nor arr[] has double number of codes.
        for (j = i-1 ; j >= 0 ; j--)
            arr.push_back(arr[j]);

        // append 0 to the first half
        for (j = 0 ; j < i ; j++)
            arr[j] = "0" + arr[j];

        // append 1 to the second half
        for (j = i ; j < 2*i ; j++)
            arr[j] = "1" + arr[j];
    }

    // print contents of arr[]
    for (i = 0 ; i < arr.size() ; i++ )
        cout << arr[i] << endl;
```

```
}

// Driver program to test above function
int main()
{
    generateGrayarr(4);
    return 0;
}
```

Output

```
000
001
011
010
110
111
101
100
```

This article is compiled by **Ravi Chandra Enaganti**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/given-a-number-n-generate-bit-patterns-from-0-to-2n-1-so-that-successive-patterns

Chapter 179

Generate n-bit Gray Codes | Set 2

Generate n-bit Gray Codes | Set 2 - GeeksforGeeks

Given a number n, generate bit patterns from 0 to 2^n-1 such that successive patterns differ by one bit.

Examples:

```
Input : n=2
Output :00 01 11 10

Input :n=3
Output :000 001 011 010 110 111 101 100
```

Another approach of Generate n-bit Gray Codes has been already discussed.

Approach is to get gray code of binary number using XOR and Right shift operation.

The first bit(MSB) of the gray code is same as the first bit(MSB) of binary number.

The second bit(from left side) of the gray code equals to XOR of first bit(MSB) and second bit(2nd MSB) of the binary number.

The third bit(from left side) of the gray code equals to XOR of the second bit(2nd MSB) and third bit(3rd MSB) and so on..

In this way we can get the gray code of corresponding binary number. So, observation is that to get i'th bit gray code, we have to do XOR operation of i'th bit and previous i-1'th bit. So by doing right shift, i-1'th bit and i'th bit come at same position and the XOR result of i'th bit and i-1'th bit will be the i'th bit gray code.That's the intuition behind right shift operation.

C++

```
 // C++ program to generate n-bit
// gray codes
#include <bits/stdc++.h>
using namespace std;

// Function to convert decimal to binary
void decimalToBinaryNumber(int x, int n)
{
    int* binaryNumber = new int(x);
    int i = 0;
    while (x > 0) {
        binaryNumber[i] = x % 2;
        x = x / 2;
        i++;
    }

    // leftmost digits are filled with 0
    for (int j = 0; j < n - i; j++)
        cout << '0';

    for (int j = i - 1; j >= 0; j--)
        cout << binaryNumber[j];
}

// Function to generate gray code
void generateGrayarr(int n)
{
    int N = 1 << n;
    for (int i = 0; i < N; i++) {

        // generate gray code of corresponding
        // binary number of integer i.
        int x = i ^ (i >> 1);

        // printing gray code
        decimalToBinaryNumber(x, n);

        cout << endl;
    }
}

// Drivers code
int main()
{
    int n = 3;
    generateGrayarr(n);
    return 0;
}
```

Java

```
 // Java program to generate
// n-bit gray codes
import java.io.*;

class GFG
{

// Function to convert
// decimal to binary
static void decimalToBinaryNumber(int x,
                                  int n)
{
    int []binaryNumber = new int[x];
    int i = 0;
    while (x > 0)
    {
        binaryNumber[i] = x % 2;
        x = x / 2;
        i++;
    }

    // leftmost digits are
    // filled with 0
    for (int j = 0; j < n - i; j++)
        System.out.print('0');

    for (int j = i - 1;
             j >= 0; j--)
        System.out.print(binaryNumber[j]);
}

// Function to generate
// gray code
static void generateGrayarr(int n)
{
    int N = 1 << n;
    for (int i = 0; i < N; i++)
    {

        // generate gray code of
        // corresponding binary
        // number of integer i.
        int x = i ^ (i >> 1);

        // printing gray code
        decimalToBinaryNumber(x, n);
```

```
        System.out.println();
    }
}

// Driver code
public static void main (String[] args)
{
    int n = 3;
    generateGrayarr(n);
}
}

// This code is contributed
// by anuj_67.
```

C#

```
 // C# program to generate
// n-bit gray codes
using System;

class GFG
{

// Function to convert
// decimal to binary
static void decimalToBinaryNumber(int x,
                                  int n)
{
    int []binaryNumber = new int[x];
    int i = 0;
    while (x > 0)
    {
        binaryNumber[i] = x % 2;
        x = x / 2;
        i++;
    }

    // leftmost digits are
    // filled with 0
    for (int j = 0; j < n - i; j++)
    Console.Write('0');

    for (int j = i - 1;
            j >= 0; j--)
        Console.Write(binaryNumber[j]);
}
```

```
// Function to generate
// gray code
static void generateGrayarr(int n)
{
    int N = 1 << n;
    for (int i = 0; i < N; i++)
    {

        // generate gray code of
        // corresponding binary
        // number of integer i.
        int x = i ^ (i >> 1);

        // printing gray code
        decimalToBinaryNumber(x, n);

        Console.WriteLine();
    }
}

// Driver code
public static void Main ()
{
    int n = 3;
    generateGrayarr(n);
}
}

// This code is contributed
// by anuj_67.
```

Output:

```
000
001
011
010
110
111
101
100
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/generate-n-bit-gray-codes-set-2/

Chapter 180

Get the position of rightmost unset bit

Get the position of rightmost unset bit - GeeksforGeeks

Given a non-negative number **n**. Find the position of rightmost unset bit in the binary representation of **n**, considering the last bit at position 1, 2nd last bit at position 2 and so on. If no **0's** are there in the binary representation of **n**. then print "-1".

Examples:

```
Input : n = 9
Output : 2
(9)10 = (1001)2
The position of rightmost unset bit in the binary
representation of 9 is 2.

Input : n = 32
Output : 1
```

Approach: Following are the steps:

1. If **n** = 0, return 1.
2. If all bits of **n** are set, return -1. Refer this post.
3. Else perform bitwise not on the given number(operation equivalent to 1's complement). Let it be **num** = ~n.
4. Get the position of rightmost set bit of **num**. This will be the position of rightmost unset bit of **n**.

C++

```
 // C++ implementation to get the position of rightmost unset bit
#include <bits/stdc++.h>

using namespace std;

// function to find the position
// of rightmost set bit
int getPosOfRightmostSetBit(int n)
{
    return log2(n&-n)+1;
}

// function to get the position of rightmost unset bit
int getPosOfRightMostUnsetBit(int n)
{
    // if n = 0, return 1
    if (n == 0)
        return 1;

    // if all bits of 'n' are set
    if ((n & (n + 1)) == 0)
        return -1;

    // position of rightmost unset bit in 'n'
    // passing ~n as argument
    return getPosOfRightmostSetBit(~n);
}

// Driver program to test above
int main()
{
    int n = 9;
    cout << getPosOfRightMostUnsetBit(n);
    return 0;
}
```

Java

```
 // Java implementation to get the
// position of rightmost unset bit

class GFG {

// function to find the position
// of rightmost set bit
static int getPosOfRightmostSetBit(int n)
{
    return (int)((Math.log10(n & -n)) / Math.log10(2)) + 1;
```

```
}

// function to get the position
// of rightmost unset bit
static int getPosOfRightMostUnsetBit(int n) {

    // if n = 0, return 1
    if (n == 0)
    return 1;

    // if all bits of 'n' are set
    if ((n & (n + 1)) == 0)
    return -1;

    // position of rightmost unset bit in 'n'
    // passing ~n as argument
    return getPosOfRightmostSetBit(~n);
}

// Driver code
public static void main(String arg[])
{
    int n = 9;
    System.out.print(getPosOfRightMostUnsetBit(n));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 implementation to get the position
# of rightmost unset bit

# import library
import math as m

# function to find the position
# of rightmost set bit
def getPosOfRightmostSetBit(n):

    return (m.log(((n & - n) + 1),2))


# function to get the position ot rightmost unset bit
def getPosOfRightMostUnsetBit(n):

    # if n = 0, return 1
```

```
    if (n == 0):
        return 1

    # if all bits of 'n' are set
    if ((n & (n + 1)) == 0):
        return -1

    # position of rightmost unset bit in 'n'
    # passing ~n as argument
    return getPosOfRightmostSetBit(~n)


# Driver program to test above
n = 13;
ans = getPosOfRightMostUnsetBit(n)

#rounding the final answer
print (round(ans))

# This code is contributed by Saloni Gupta.
```

C#

```
 // C# implementation to get the
// position of rightmost unset bit
using System;

class GFG
{
    // function to find the position
    // of rightmost set bit
    static int getPosOfRightmostSetBit(int n)
    {
        return (int)((Math.Log10(n & -n)) / Math.Log10(2)) + 1;
    }

    // function to get the position
    // of rightmost unset bit
    static int getPosOfRightMostUnsetBit(int n) {

        // if n = 0, return 1
        if (n == 0)
        return 1;

        // if all bits of 'n' are set
        if ((n & (n + 1)) == 0)
        return -1;
```

```
        // position of rightmost unset bit in 'n'
        // passing ~n as argument
        return getPosOfRightmostSetBit(~n);
    }

    // Driver code
    public static void Main()
    {
        int n = 9;
        Console.Write(getPosOfRightMostUnsetBit(n));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP implementation to get the
// position of rightmost unset bit

// function to find the position
// of rightmost set bit
function getPosOfRightmostSetBit( $n)
{
    return ceil(log($n &- $n) + 1);
}

// function to get the position
// of rightmost unset bit
function getPosOfRightMostUnsetBit( $n)
{
    // if n = 0, return 1
    if ($n == 0)
        return 1;

    // if all bits of 'n' are set
    if (($n & ($n + 1)) == 0)
        return -1;

    // position of rightmost unset bit in 'n'
    // passing ~n as argument
    return getPosOfRightmostSetBit(~$n);
}

    // Driver Code
    $n = 9;
    echo getPosOfRightMostUnsetBit($n);
```

```
// This code is contributed by anuj_67.
?>
```

Output:

```
2
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/get-position-rightmost-unset-bit/

Chapter 181

Given a set, find XOR of the XOR's of all subsets.

Given a set, find XOR of the XOR's of all subsets. - GeeksforGeeks

The question is to find XOR of the XOR's of all subsets. i.e if the set is {1,2,3}. All subsets are : [{1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}]. Find the XOR of each of the subset and then find the XOR of every subset result.

We strongly recommend you to minimize your browser and try this yourself first.

This is a very simple question to solve if we get the first step (and only step) right. The solution is XOR is always 0 when n > 1 and Set[0] when n is 1.

C++

```
 // C++ program to find XOR of XOR's of all subsets
#include <bits/stdc++.h>
using namespace std;

// Returns XOR of all XOR's of given subset
int findXOR(int Set[], int n)
{
    // XOR is 1 only when n is 1, else 0
    if (n == 1)
       return Set[0];
    else
       return 0;
}

// Driver program
int main()
```

```
{
    int Set[] = {1, 2, 3};
    int n = sizeof(Set)/sizeof(Set[0]);
    cout << "XOR of XOR's of all subsets is "
         << findXOR(Set, n);
    return 0;
}
```

Java

```
 // Java program to find XOR of
// XOR's of all subsets
import java.util.*;

class GFG {

// Returns XOR of all XOR's of given subset
static int findXOR(int Set[], int n) {

    // XOR is 1 only when n is 1, else 0
    if (n == 1)
    return Set[0];
    else
    return 0;
}

// Driver code
public static void main(String arg[])
{
    int Set[] = {1, 2, 3};
    int n = Set.length;
    System.out.print("XOR of XOR's of all subsets is " +
                                          findXOR(Set, n));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python program to find
# XOR of XOR's of all subsets

# Returns XOR of all
# XOR's of given subset
def findXOR(Set, n):
```

```
    # XOR is 1 only when
    # n is 1, else 0
    if (n == 1):
         return Set[0]
    else:
         return 0

# Driver code

Set = [1, 2, 3]
n = len(Set)

print("XOR of XOR's of all subsets is ",
         findXOR(Set, n));

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to find XOR of
// XOR's of all subsets
using System;

class GFG {

// Returns XOR of all
// XOR's of given subset
static int findXOR(int []Set, int n)
{

    // XOR is 1 only when n
    // is 1, else 0
    if (n == 1)
    return Set[0];
    else
    return 0;
}

// Driver code
public static void Main()
{
    int []Set = {1, 2, 3};
    int n = Set.Length;
    Console.Write("XOR of XOR's of all subsets is " +
                                     findXOR(Set, n));
}
}
```

```
// This code is contributed by nitin mittal
```

PHP

```
 <?php
// PHP program to find XOR
// of XOR's of all subsets

// Returns XOR of all
// XOR's of given subset
function findXOR($Set, $n)
{

    // XOR is 1 only when
    // n is 1, else 0
    if ($n == 1)
        return $Set[0];
    else
        return 0;
}

    // Driver Code
    $Set = array(1, 2, 3);
    $n = count($Set);
    echo "XOR of XOR's of all subsets is "
         , findXOR($Set, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
XOR of XOR's of all subsets is 0
```

Related Problem :
Sum of XOR of all possible subsets

How does this work?
The logic goes simple. Let us consider n'th element, it can be included in all subsets of remaining (n-1) elements. The number of subsets for (n-1) elements is equal to $2^{(n-1)}$ which is always even when n>1. Thus, in the XOR result, every element is included even number of times and XOR of even occurrences of any number is 0.

This article is contributed by **Ekta Goel**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/given-a-set-find-xor-of-the-xors-of-all-subsets/

Chapter 182

Gray to Binary and Binary to Gray conversion

Gray to Binary and Binary to Gray conversion - GeeksforGeeks

Binary Numbers is default way to store numbers, but in many applications binary numbers are difficult to use and a variation of binary numbers is needed. This is where Gray codes are very useful.
Gray code has property that **two successive numbers differ in only one bit** because of this property gray code does the cycling through various states with minimal effort and used in K-maps, error correction, communication etc.

How to generate n bit Gray Codes?

```
Following is 2-bit sequence (n = 2)
  00 01 11 10
Following is 3-bit sequence (n = 3)
  000 001 011 010 110 111 101 100
And Following is 4-bit sequence (n = 4)
  0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111
  1110 1010 1011 1001 1000
```

n-bit Gray Codes can be generated from list of (n-1)-bit Gray codes using following steps.

Let the list of (n-1)-bit Gray codes be L1. Create another list L2 which is reverse of L1.

Modify the list L1 by prefixing a '0' in all codes of L1.

Modify the list L2 by prefixing a '1' in all codes of L2.

Concatenate L1 and L2. The concatenated list is required list of n-bit Gray codes.

Please refer Generate n-bit Gray Codesfor detailed program.

How to Convert Binary To Gray and Vice Versa?

```
Binary : 0011
Gray   : 0010

Binary : 01001
Gray   : 01101
```

In computer science many a times we need to convert binary code to gray code and vice versa. This conversion can be done by applying following rules :
Binary to Gray conversion :

1. The Most Significant Bit (MSB) of the gray code is always equal to the MSB of the given binary code.
2. Other bits of the output gray code can be obtained by XORing binary code bit at that index and previous index.

Gray to binary conversion :

1. The Most Significant Bit (MSB) of the binary code is always equal to the MSB of the given binary number.
2. Other bits of the output binary code can be obtained by checking gray code bit at that index. If current gray code bit is 0, then copy previous binary code bit, else copy invert of previous binary code bit.

Below is the implementation of above steps.

C++

```
 // C++ program for Binary To Gray
// and Gray to Binary conversion
#include <iostream>
using namespace std;

// Helper function to xor two characters
char xor_c(char a, char b) { return (a == b) ? '0' : '1'; }

// Helper function to flip the bit
char flip(char c) { return (c == '0') ? '1' : '0'; }

// function to convert binary string
// to gray string
string binarytoGray(string binary)
{
    string gray = "";

    // MSB of gray code is same as binary code
```

```
    gray += binary[0];

    // Compute remaining bits, next bit is comuted by
    // doing XOR of previous and current in Binary
    for (int i = 1; i < binary.length(); i++) {
        // Concatenate XOR of previous bit
        // with current bit
        gray += xor_c(binary[i - 1], binary[i]);
    }

    return gray;
}

// function to convert gray code string
// to binary string
string graytoBinary(string gray)
{
    string binary = "";

    // MSB of binary code is same as gray code
    binary += gray[0];

    // Compute remaining bits
    for (int i = 1; i < gray.length(); i++) {
        // If current bit is 0, concatenate
        // previous bit
        if (gray[i] == '0')
            binary += binary[i - 1];

        // Else, concatenate invert of
        // previous bit
        else
            binary += flip(binary[i - 1]);
    }

    return binary;
}

// Driver program to test above functions
int main()
{
    string binary = "01001";
    cout << "Gray code of " << binary << " is "
         << binarytoGray(binary) << endl;

    string gray = "01101";
    cout << "Binary code of " << gray << " is "
         << graytoBinary(gray) << endl;
```

```
    return 0;
}
```

Java

```
 // Java program for Binary To Gray
// and Gray to Binary conversion
import java.io.*;
class code_conversion {
    // Helper function to xor
    // two characters
    char xor_c(char a, char b)
    {
        return (a == b) ? '0' : '1';
    }

    // Helper function to flip the bit
    char flip(char c)
    {
        return (c == '0') ? '1' : '0';
    }

    // function to convert binary
    // string to gray string
    String binarytoGray(String binary)
    {
        String gray = "";

        // MSB of gray code is same
        // as binary code
        gray += binary.charAt(0);

        // Compute remaining bits, next bit is
        // comuted by doing XOR of previous
        // and current in Binary
        for (int i = 1; i < binary.length(); i++) {
            // Concatenate XOR of previous bit
            // with current bit
            gray += xor_c(binary.charAt(i - 1),
                          binary.charAt(i));
        }

        return gray;
    }
    // function to convert gray code
    // string to binary string
    String graytoBinary(String gray)
    {
```

```
        String binary = "";

        // MSB of binary code is same
        // as gray code
        binary += gray.charAt(0);

        // Compute remaining bits
        for (int i = 1; i < gray.length(); i++) {
            // If current bit is 0,
            // concatenate previous bit
            if (gray.charAt(i) == '0')
                binary += binary.charAt(i - 1);

            // Else, concatenate invert of
            // previous bit
            else
                binary += flip(binary.charAt(i - 1));
        }

        return binary;
    }

    // Driver program to test above functions
    public static void main(String args[])
        throws IOException
    {
        code_conversion ob = new code_conversion();
        String binary = "01001";
        System.out.println("Gray code of " + binary + " is " + ob.binarytoGray(binary));

        String gray = "01101";
        System.out.println("Binary code of " + gray + " is " + ob.graytoBinary(gray));
    }
}

// This code is contributed by Anshika Goyal.
```

C#

```
 // C# program for Binary To Gray
// and Gray to Binary conversion.
using System;

class GFG {

    // Helper function to xor
    // two characters
    static char xor_c(char a, char b)
```

```
{
    return (a == b) ? '0' : '1';
}

// Helper function to flip the bit
static char flip(char c)
{
    return (c == '0') ? '1' : '0';
}

// function to convert binary
// string to gray string
static String binarytoGray(String binary)
{
    String gray = "";

    // MSB of gray code is same
    // as binary code
    gray += binary[0];

    // Compute remaining bits, next
    // bit is comuted by doing XOR
    // of previous and current in
    // Binary
    for (int i = 1; i < binary.Length; i++)
    {

        // Concatenate XOR of previous
        // bit with current bit
        gray += xor_c(binary[i - 1],
                              binary[i]);
    }

    return gray;
}

// function to convert gray code
// string to binary string
static String graytoBinary(String gray)
{

    String binary = "";

    // MSB of binary code is same
    // as gray code
    binary += gray[0];

    // Compute remaining bits
```

```
        for (int i = 1; i < gray.Length; i++)
        {

            // If current bit is 0,
            // concatenate previous bit
            if (gray[i] == '0')
                binary += binary[i - 1];

            // Else, concatenate invert of
            // previous bit
            else
                binary += flip(binary[i - 1]);
        }

        return binary;
    }

    // Driver program to test above
    // functions
    public static void Main()

    {
        String binary = "01001";
        Console.WriteLine("Gray code of "
                        + binary + " is "
                 + binarytoGray(binary));

        String gray = "01101";
        Console.Write("Binary code of "
                        + gray + " is "
                 + graytoBinary(gray));
    }
}

// This code is contributed by nitin mittal.
```

Output:

```
Gray code of 01001 is 01101
Binary code of 01101 is 01001
```

This article is contributed by Utkarsh Trivedi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/gray-to-binary-and-binary-to-gray-conversion/

Chapter 183

Highest power of 2 less than or equal to given number

Highest power of 2 less than or equal to given number - GeeksforGeeks

Given a number n, find the highest power of 2 that is smaller than or equal to n.

Examples :

```
Input : n = 10
Output : 8

Input : n = 19
Output : 16

Input : n = 32
Output : 32
```

A **simple solution** is to start checking from n and keep decrementing until we find a power of 2.

C++

```
 // C++ program to find highest power of 2 smaller
// than or equal to n.
#include<bits/stdc++.h>
using namespace std;

int highestPowerof2(int n)
{
    int res = 0;
```

```
    for (int i=n; i>=1; i--)
    {
        // If i is a power of 2
        if ((i & (i-1)) == 0)
        {
            res = i;
            break;
        }
    }
    return res;
}

// Driver code
int main()
{
    int n = 10;
    cout << highestPowerof2(n);
    return 0;
}
```

Java

```
 // Java code to find highest power
// of 2 smaller than or equal to n.
class GFG
{
   static int highestPowerof2(int n)
   {
     int res = 0;
     for (int i = n; i >= 1; i--)
        {
         // If i is a power of 2
          if ((i & (i - 1)) == 0)
             {
                  res = i;
               break;
             }
        }
     return res;
   }

   // Driver code
   public static void main(String[] args)
   {
      int n = 10;
      System.out.println(highestPowerof2(n));
   }
}
```

```
// This code is contributed by prerna saini.
```

C#

```
 // C# code to find highest power
// of 2 smaller than or equal to n.
using System;

class GFG
{
public static int highestPowerof2(int n)
{
    int res = 0;
    for (int i = n; i >= 1; i--)
        {
        // If i is a power of 2
        if ((i & (i - 1)) == 0)
            {
                res = i;
                break;
            }
        }
    return res;
}

    // Driver Code
    static public void Main ()
    {
        int n = 10;
        Console.WriteLine(highestPowerof2(n));
    }
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP program to find highest
// power of 2 smaller than or
// equal to n.
function highestPowerof2($n)
{
    $res = 0;
    for ($i = $n; $i >= 1; $i--)
    {
```

```
        // If i is a power of 2
        if (($i & ($i - 1)) == 0)
        {
            $res = $i;
            break;
        }
    }
    return $res;
}

// Driver code
$n = 10;
echo highestPowerof2($n);

// This code is contributed by m_kit
?>
```

Output :

```
8
```

Time complexity : O(n). In worst case, the loop runs floor(n/2) times. The worst case happens when n is of the form $2^x - 1$.

An **efficient solution** is to use bitwise left shift operator to find all powers of 2 starting from 1. For every power check if it is smaller than or equal to n or not. Below is C++ implementation of the idea.

C++

```
 // C++ program to find highest power of 2 smaller
// than or equal to n.
#include<bits/stdc++.h>
using namespace std;

int highestPowerof2(unsigned int n)
{
    // Invalid input
    if (n < 1)
        return 0;

    int res = 1;

    // Try all powers starting from 2^1
    for (int i=0; i<8*sizeof(unsigned int); i++)
    {
```

```
        int curr = 1 << i;

        // If current power is more than n, break
        if (curr > n)
           break;

        res = curr;
    }

    return res;
}

// Driver code
int main()
{
    int n = 10;
    cout << highestPowerof2(n);
    return 0;
}
```

Java

```
 // Java program to find
// highest power of 2 smaller
// than or equal to n.
import java.io.*;

class GFG
{
static int highestPowerof2(int n)
{
    // Invalid input
    if (n < 1)
        return 0;

    int res = 1;

    // Try all powers
    // starting from 2^1
    for (int i = 0; i < 8 ; i++)
    {
        int curr = 1 << i;

        // If current power is
        // more than n, break
        if (curr > n)
        break;
```

```
        res = curr;
    }

    return res;
}

// Driver code
public static void main(String[] args)
{
    int n = 10;
    System.out.println(highestPowerof2(n));
}
}

// This code is contributed aj_36
```

C#

```
 // C# program to find
// highest power of 2 smaller
// than or equal to n.
using System;

class GFG
{
static int highestPowerof2(int n)
{
    // Invalid input
    if (n < 1)
        return 0;

    int res = 1;

    // Try all powers
    // starting from 2^1
    for (int i = 0; i < 8 ; i++)
    {
        int curr = 1 << i;

        // If current power is
        // more than n, break
        if (curr > n)
        break;

        res = curr;
    }

    return res;
```

```
}

// Driver code
static public void Main ()
{
    int n = 10;
    Console.WriteLine(highestPowerof2(n));
}
}

// This code is contributed ajit
```

PHP

```
 <?php
// PHP program to find highest
// power of 2 smaller
// than or equal to n.

function highestPowerof2($n)
{
    // Invalid input
    if ($n < 1)
        return 0;

    $res = 1;

    // Try all powers starting
    // from 2^1
    for ($i = 0; $i < 8 ; $i++)
    {
        $curr = 1 << $i;

        // If current power is
        // more than n, break
        if ($curr > $n)
        break;

        $res = $curr;
    }

    return $res;
}

// Driver code
$n = 10;
echo highestPowerof2($n);
```

```
// This code is contributed
// by m_kit
?>
```

Output :

```
8
```

A Solution using Log
Thanks to Anshuman Jha for suggesting this solution.

C++

```
 // C++ program to find highest power of 2 smaller
// than or equal to n.
#include<bits/stdc++.h>
using namespace std;

int highestPowerof2(int n)
{
   int p = (int)log2(n);
   return (int)pow(2, p);
}

// Driver code
int main()
{
    int n = 10;
    cout << highestPowerof2(n);
    return 0;
}
```

Java

```
 // Java program to find
// highest power of 2
// smaller than or equal to n.
import java.io.*;

class GFG
{
static int highestPowerof2(int n)
{

    int p = (int)(Math.log(n) /
                  Math.log(2));
    return (int)Math.pow(2, p);
```

```
}

// Driver code
public static void main (String[] args)
{
    int n = 10;
    System.out.println(highestPowerof2(n));
}
}

// This code is contributed
// by m_kit
```

C#

```
 // C# program to find
// highest power of 2
// smaller than or equal to n.
using System;

class GFG
{
static int highestPowerof2(int n)
{
    int p = (int)(Math.Log(n) /
                  Math.Log(2));
    return (int)Math.Pow(2, p);
}

// Driver code
static public void Main ()
{
    int n = 10;
    Console.WriteLine(highestPowerof2(n));
}
}

// This code is contributed
// by ajit
```

PHP

```
 <?php
// PHP program to find highest
// power of 2 smaller than or
// equal to n.
function highestPowerof2($n)
```

```
{
    $p = (int)log($n, 2);
    return (int)pow(2, $p);
}

// Driver code
$n = 10;
echo highestPowerof2($n);

// This code is contributed by ajit
?>
```

Output :

```
8
```

Application Problem:
Some people are standing in a queue. A selection process follows a rule where people standing on even positions are selected. Of the selected people a queue is formed and again out of these only people on even position are selected. This continues until we are left with one person. Find out the position of that person in the original queue.
Print the position(original queue) of that person who is left.
Examples :

```
Input: n = 10
Output:8
Explanation :
1 2 3 4 5 6 7 8 9 10  ===>Given queue
    2 4 6 8 10
       4 8
        8

Input: n = 17
Input: 16
Explanation :
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17  ===>Given queue
        2 4 6 8 10 12 14 16
            4 8 12 16
              8 16
               16
```

Related Article :
Power of 2 greater than or equal to a given number.

Improved By : jit_t

Source

https://www.geeksforgeeks.org/highest-power-2-less-equal-given-number/

Chapter 184

How to swap two bits in a given integer?

How to swap two bits in a given integer? - GeeksforGeeks

Given an integer n and two bit positions p1 and p2 inside it, swap bits at the given positions. The given positions are from least significant bit (lsb). For example, the position for lsb is 0.

Examples:

```
Input: n = 28, p1 = 0, p2 = 3
Output: 21
28 in binary is 11100.  If we swap 0'th and 3rd digits,
we get 10101 which is 21 in decimal.

Input: n = 20, p1 = 2, p2 = 3
Output: 24
```

We strongly recommend you to minimize your browser and try this yourself first.

The idea is to first find the bits, then use XOR based swapping concept, i..e., to swap two numbers 'x' and 'y', we do x = x ^ y, y = y ^ x and x = x ^ y.

Below is C implementation.

```
 // C program to swap bits in an intger
#include<stdio.h>

// This function swaps bit at positions p1 and p2 in an integer n
int swapBits(unsigned int n, unsigned int p1, unsigned int p2)
{
```

```
    /* Move p1'th to rightmost side */
    unsigned int bit1 =  (n >> p1) & 1;

    /* Move p2'th to rightmost side */
    unsigned int bit2 =  (n >> p2) & 1;

    /* XOR the two bits */
    unsigned int x = (bit1 ^ bit2);

    /* Put the xor bit back to their original positions */
    x = (x << p1) | (x << p2);

    /* XOR 'x' with the original number so that the
       two sets are swapped */
    unsigned int result = n ^ x;
}

/* Drier program to test above function*/
int main()
{
    int res =  swapBits(28, 0, 3);
    printf("Result = %d ", res);
    return 0;
}
```

Output:

```
Result = 21
```

Improved By : jit_t

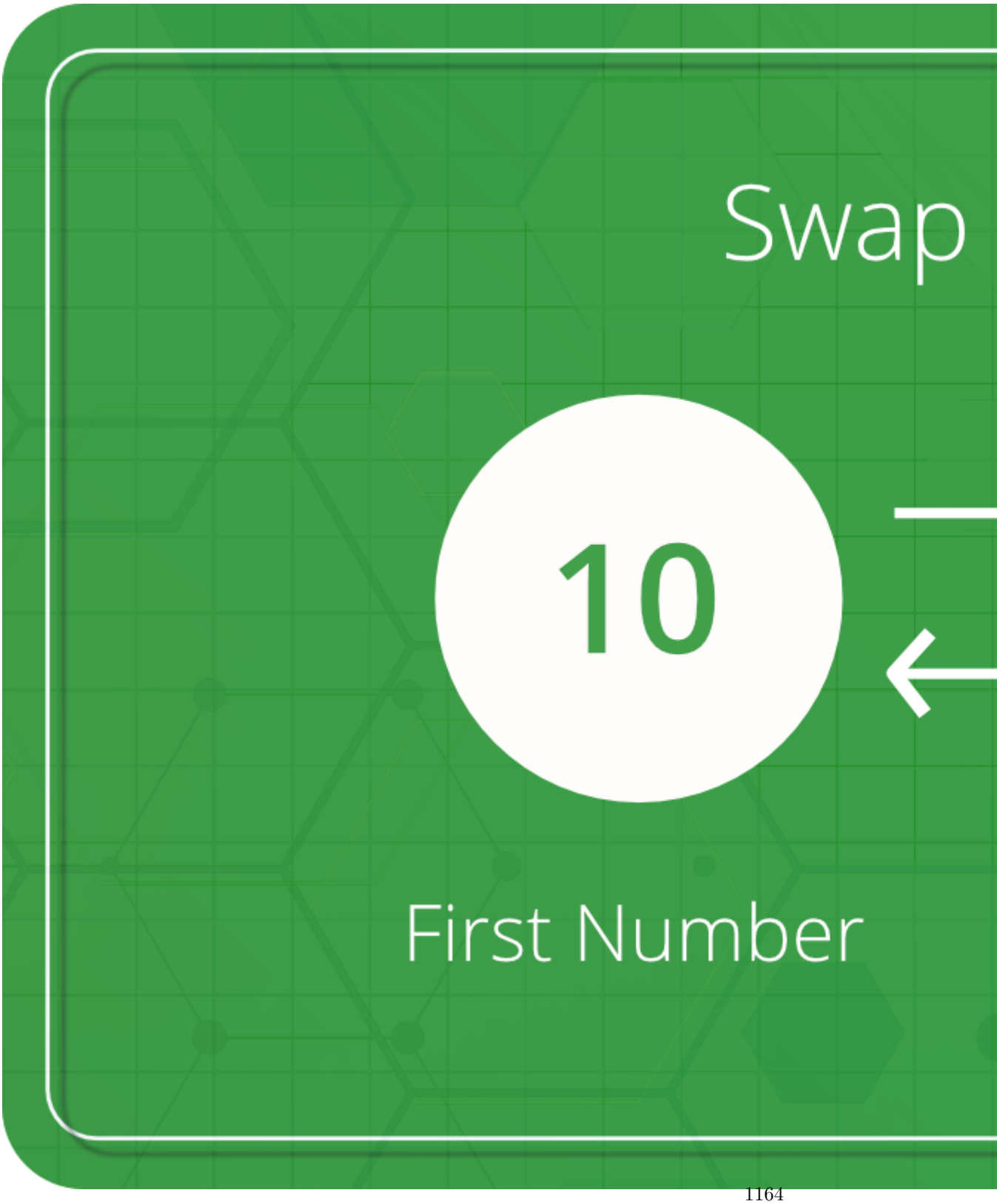
Source

https://www.geeksforgeeks.org/how-to-swap-two-bits-in-a-given-integer/

Chapter 185

How to swap two numbers without using a temporary variable?

How to swap two numbers without using a temporary variable? - GeeksforGeeks
Given two variables, x and y, swap two variables without using a third variable.


Swap
10
First Number

Method 1 (Using Arithmetic Operators)
The idea is to get sum in one of the two given numbers. The numbers can then be swapped using the sum and subtraction from sum.

C

```
 #include <stdio.h>
int main()
{
  int x = 10, y = 5;

  // Code to swap 'x' and 'y'
  x = x + y;  // x now becomes 15
  y = x - y;  // y becomes 10
  x = x - y;  // x becomes 5

  printf("After Swapping: x = %d, y = %d", x, y);

  return 0;
}
```

Java

```
 // Program to swap two numbers  without
// using temporary variable
import java.*;

class Geeks {

    public static void main(String a[])
    {
        int x = 10;
        int y = 5;
        x = x + y;
        y = x - y;
        x = x - y;
        System.out.println("After swaping:"
             + " x = " + x + ", y = " + y);
    }
}

// This code is contributed by Sam007
```

Python 3

```
 x = 10
```

```
y = 5

# Code to swap 'x' and 'y'

# x now becomes 15
x = x + y

# y becomes 10
y = x - y

# x becomes 5
x = x - y
print("After Swapping: x =",x ," y =", y);

# This code is contributed
# by Sumit Sudhakar
```

C#

```
 // Program to swap two numbers  without
// using temporary variable
using System;

class GFG {
public static void Main()
    {
        int x = 10;
        int y = 5;
        Console.WriteLine("Before swap:");
        Console.WriteLine("x value: " + x);
        Console.WriteLine("y value: " + y);
        x = x + y;
        y = x - y;
        x = x - y;
        Console.WriteLine("After swap:");
        Console.WriteLine("x value: " + x);
        Console.WriteLine("y value: " + y);
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP Program to swap two
// numbers without using
```

```
// temporary variable
$x = 10; $y = 5;

// Code to swap 'x' and 'y'
$x = $x + $y; // x now becomes 15
$y = $x - $y; // y becomes 10
$x = $x - $y; // x becomes 5

echo "After Swapping: x = ",
       $x, "," , "y = ", $y;

// This code is contributed by m_kit
?>
```

Output :

```
After Swapping: x = 5, y = 10
```

Multiplication and division can also be used for swapping.

C

```
 #include <stdio.h>
int main()
{
  int x = 10, y = 5;

  // Code to swap 'x' and 'y'
  x = x * y;  // x now becomes 50
  y = x / y;  // y becomes 10
  x = x / y;  // x becomes 5

  printf("After Swapping: x = %d, y = %d", x, y);

  return 0;
}
```

Java

```
 // Java Program to swap two numbers
// without using temporary variable
import java.io.*;

class GFG
{
    public static void main (String[] args)
    {
```

```
        int x = 10;
        int y = 5;

        // Code to swap 'x' and 'y'
        x = x * y; // x now becomes 50
        y = x / y; // y becomes 10
        x = x / y; // x becomes 5

        System.out.println("After swaping:" +
                                " x = " + x +
                                ", y = " + y);
    }
}

// This code is contributed by ajit
```

Python3

```
 # Python3 program to
# swap two numbers
# without using
# temporary variable
x = 10
y = 5

# code to swap
# 'x' and 'y'

# x now becomes 50
x = x * y

# y becomes 10
y = x // y;

# x becomes 5
x = x // y;

print("After Swapping: x =",
              x ," y =", y);

# This code is contributed
# by @ajit
```

C#

```
 // C# Program to swap two
// numbers without using
```

```
// temporary variable
using System;

class GFG
{
    static public void Main ()
    {
        int x = 10;
        int y = 5;

        // Code to swap 'x' and 'y'
        x = x * y; // x now becomes 50
        y = x / y; // y becomes 10
        x = x / y; // x becomes 5

        Console.WriteLine("After swaping:" +
                                " x = " + x +
                               ", y = " + y);
    }
}

// This code is contributed by ajit.
```

PHP

```
 <?php
// Driver code
    $x = 10;
    $y = 5;

// Code to swap 'x' and 'y'
    $x = $x * $y; // x now becomes 50
    $y = $x / $y; // y becomes 10
    $x = $x / $y; // x becomes 5

echo "After Swapping: x = ", $x,
                " ", "y = ", $y;

// This code is contributed by m_kit
?>
```

Output :

```
After Swapping: x = 5, y = 10
```

Method 2 (Using Bitwise XOR)
The bitwise XOR operator can be used to swap two variables. The XOR of two numbers

x and y returns a number which has all the bits as 1 wherever bits of x and y differ. For example XOR of 10 (In Binary 1010) and 5 (In Binary 0101) is 1111 and XOR of 7 (0111) and 5 (0101) is (0010).

C

```
 #include <stdio.h>
int main()
{
  int x = 10, y = 5;

  // Code to swap 'x' (1010) and 'y' (0101)
  x = x ^ y;  // x now becomes 15 (1111)
  y = x ^ y;  // y becomes 10 (1010)
  x = x ^ y;  // x becomes 5 (0101)

  printf("After Swapping: x = %d, y = %d", x, y);

  return 0;
}
```

Java

```
 import java.*;

public class GFG {

    public static void main(String a[]){
        int x = 10;
        int y = 5;

        // Code to swap 'x' (1010) and 'y' (0101)
        x = x ^ y; // x now becomes 15 (1111)
        y = x ^ y; // y becomes 10 (1010)
        x = x ^ y; // x becomes 5 (0101)

        System.out.println("After swap: x = "
                           + x + ", y = " + y);
    }
}

// This code is contributed by Sam007.
```

Python3

```
 # Python 3 code to swap using XOR
```

```
x = 10
y = 5

# Code to swap 'x' and 'y'
x = x ^ y; # x now becomes 15 (1111)
y = x ^ y; # y becomes 10 (1010)
x = x ^ y; # x becomes 5 (0101)

print ("After Swapping: x = ",x ," y =", y)

# This code is contributed by
# Sumit Sudhakar
```

C#

```
 // C# program to swap using XOR
using System;

class GFG
{
    public static void Main()
    {
        int x = 10;
        int y = 5;

        // Code to swap 'x' (1010)
        // and 'y' (0101)

        // x now becomes 15 (1111)
        x = x ^ y;

        // y becomes 10 (1010)
        y = x ^ y;

        // x becomes 5 (0101)
        x = x ^ y;

    Console.WriteLine("After swap: x = " +
                        x + ", y = " + y);
    }
}

// This code is contributed by ajit
```

PHP

```
 <?php
```

```
// Driver Code
$x = 10;
$y = 5;

// Code to swap 'x' (1010)
// and 'y' (0101)

// x now becomes 15 (1111)
$x = $x ^ $y;

// y becomes 10 (1010)
$y = $x ^ $y;

// x becomes 5 (0101)
$x = $x ^ $y;

echo "After Swapping: x = ", $x,
                ",", "y = ", $y;

// This code is contributed by aj_36
?>
```

Output :

```
After Swapping: x = 5, y = 10
```

Problems with above methods

1) The multiplication and division based approach doesn' work if one of the numbers is 0 as the product becomes 0 irrespective of the other number.

2) Both Arithmetic solutions may cause arithmetic overflow. If x and y are too large, addition and multiplication may go out of integer range.

3) When we use pointers to variable and make a function swap, all of the above methods fail when both pointers point to the same variable. Let's take a look what will happen in this case if both are pointing to the same variable.

// Bitwise XOR based method
x = x ^ x; // x becomes 0
x = x ^ x; // x remains 0
x = x ^ x; // x remains 0

// Arithmetic based method
x = x + x; // x becomes 2x
x = x – x; // x becomes 0
x = x – x; // x remains 0

Let us see the following program.

C

```
 #include <stdio.h>
void swap(int *xp, int *yp)
{
    *xp = *xp ^ *yp;
    *yp = *xp ^ *yp;
    *xp = *xp ^ *yp;
}

int main()
{
  int x = 10;
  swap(&x, &x);
  printf("After swap(&x, &x): x = %d", x);
  return 0;
}
```

Output :

```
After swap(&x, &x): x = 0
```

Swapping a variable with itself may needed in many standard algorithms. For example see thisimplementation of QuickSortwhere we may swap a variable with itself. The above problem can be avoided by putting a condition before the swapping.

C

```
 #include <stdio.h>
void swap(int *xp, int *yp)
{
    if (xp == yp) // Check if the two addresses are same
      return;
    *xp = *xp + *yp;
    *yp = *xp - *yp;
    *xp = *xp - *yp;
}
int main()
{
  int x = 10;
  swap(&x, &x);
  printf("After swap(&x, &x): x = %d", x);
  return 0;
}
```

Output :

```
After swap(&x, &x): x = 10
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/swap-two-numbers-without-using-temporary-variable/

Chapter 186

How to turn off a particular bit in a number?

How to turn off a particular bit in a number? - GeeksforGeeks

Difficulty Level: Rookie

Given a number n and a value k, turn of the k'th bit in n.

Examples:

```
Input:  n = 15, k = 1
Output: 14

Input:  n = 15, k = 2
Output: 13

Input:  n = 15, k = 3
Output: 11

Input:  n = 15, k = 4
Output: 7

Input:  n = 15, k >= 5
Output: 15
```

The idea is to use bitwise <<, & and ~ operators. Using expression "*~(1 << (k − 1))*", we get a number which has all bits set, except the k'th bit. If we do bitwise & of this expression with n, we get a number which has all bits same as n except the k'th bit which is 0.

Below is the implementation of above idea.

C++

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

// Returns a number that has all bits same as n
// except the k'th bit which is made 0
int turnOffK(int n, int k)
{
    // k must be greater than 0
    if (k <= 0) return n;

    // Do & of n with a number with all set bits except
    // the k'th bit
    return (n & ~(1 << (k - 1)));
}

// Driver program to test above function
int main()
{
    int n = 15;
    int k = 4;
    cout << turnOffK(n, k);
    return 0;
}
```

Java

```
 // Java program to turn off a particular bit in a number
import java.io.*;

class TurnOff
{
    // Function to returns a number that has all bits same as n
    // except the k'th bit which is made 0
    static int turnOffK(int n, int k)
    {
        // k must be greater than 0
        if (k <= 0)
            return n;

        // Do & of n with a number with all set bits except
        // the k'th bit
        return (n & ~(1 << (k - 1)));
    }

    // Driver program
    public static void main (String[] args)
    {
        int n = 15;
```

```
        int k = 4;
        System.out.println(turnOffK(n, k));
    }
}
// Contributed by Pramod Kumar
```

Python3

```
 # Returns a number that
# has all bits same as n
# except the k'th bit
# which is made 0

def turnOffK(n,k):

    # k must be greater than 0
    if (k <= 0):
        return n

    # Do & of n with a number
    # with all set bits except
    # the k'th bit
    return (n & ~(1 << (k - 1)))


# Driver code
n = 15
k = 4
print(turnOffK(n, k))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to turn off a
// particular bit in a number
using System;

class GFG
{

    // Function to returns a number
    // that has all bits same as n
    // except the k'th bit which is
    // made 0
    static int turnOffK(int n, int k)
```

```
    {
        // k must be greater than 0
        if (k <= 0)
            return n;

        // Do & of n with a number
        // with all set bits except
        // the k'th bit
        return (n & ~ (1 << (k - 1)));
    }

    // Driver Code
    public static void Main ()
    {
        int n = 15;
        int k = 4;
        Console.Write(turnOffK(n, k));
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP program to turn off a
// particular bit in a number

// Returns a number that has
// all bits same as n except
// the k'th bit which is made 0
function turnOffK($n, $k)
{

    // k must be greater than 0
    if ($k <= 0)
        return $n;

    // Do & of n with a number
    // with all set bits except
    // the k'th bit
    return ($n & ~(1 << ($k - 1)));
}

// Driver Code
$n = 15;
$k = 4;
echo turnOffK($n, $k);
```

```
// This code is contributed by nitin mittal
?>
```

Output:

```
7
```

Exercise: Write a function turnOnK() that turns the k'th bit on.

This article is contributed by **Rahul Jain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/how-to-turn-off-a-particular-bit-in-a-number/

Chapter 187

How to turn on a particular bit in a number?

How to turn on a particular bit in a number? - GeeksforGeeks

Given a number n and a value k, turn on the k'th bit in n.

Examples:

```
Input:  n = 4, k = 2
Output: 6

Input:  n = 3, k = 3
Output: 7

Input:  n = 64, k = 4
Output: 72

Input:  n = 64, k = 5
Output: 80
```

The idea is to use bitwise << and | operators. Using expression "(1 << (k – 1))", we get a number which has all bits unset, except the k'th bit. If we do bitwise | of this expression with n, we get a number which has all bits same as n except the k'th bit which is 1.

Below is the implementation of above idea.

C++

```
 // CPP program to turn on a particular bit
#include <iostream>
using namespace std;
```

```
// Returns a number that has all bits same as n
// except the k'th bit which is made 1
int turnOnK(int n, int k)
{
    // k must be greater than 0
    if (k <= 0)
        return n;

    // Do | of n with a number with all
    // unset bits except the k'th bit
    return (n | (1 << (k - 1)));
}

// Driver program to test above function
int main()
{
    int n = 4;
    int k = 2;
    cout << turnOnK(n, k);
    return 0;
}
```

Java

```
 // Java program to turn on a particular
// bit
class GFG {

    // Returns a number that has all
    // bits same as n except the k'th
    // bit which is made 1
    static int turnOnK(int n, int k)
    {

        // k must be greater than 0
        if (k <= 0)
            return n;

        // Do | of n with a number with
        // all unset bits except the
        // k'th bit
        return (n | (1 << (k - 1)));
    }

    // Driver program to test above
    // function
    public static void main(String [] args)
    {
```

```
        int n = 4;
        int k = 2;
        System.out.print(turnOnK(n, k));
    }
}

// This code is contributed by Smitha
```

Python 3

```
 # Python 3 program to turn on a
# particular bit

# Returns a number that has all
# bits same as n except the k'th
# bit which is made 1
def turnOnK(n, k):

    # k must be greater than 0
    if (k <= 0):
        return n

    # Do | of n with a number
    # with all unset bits except
    # the k'th bit
    return (n | (1 << (k - 1)))

# Driver program to test above
# function
n = 4
k = 2
print(turnOnK(n, k))

# This code is contributed by
# Smitha
```

C#

```
 // C# program to turn on a particular
// bit
using System;

class GFG {

    // Returns a number that has all
    // bits same as n except the k'th
    // bit which is made 1
```

```
    static int turnOnK(int n, int k)
    {

        // k must be greater than 0
        if (k <= 0)
            return n;

        // Do | of n with a number
        // with all unset bits except
        // the k'th bit
        return (n | (1 << (k - 1)));
    }

    // Driver program to test above
    // function
    public static void Main()
    {
        int n = 4;
        int k = 2;
        Console.Write(turnOnK(n, k));
    }
}

// This code is contributed by Smitha
```

PHP

```
 <?php
// PHP program to turn on a particular bit

// Returns a number that has
// all bits same as n except
// the k'th bit which is made 1
function turnOnK($n, $k)
{

    // k must be greater than 0
    if ($k <= 0)
        return $n;

    // Do | of n with a number with all
    // unset bits except the k'th bit
    return ($n | (1 << ($k - 1)));
}

    // Driver Code
    $n = 4;
    $k = 2;
```

```
    echo turnOnK($n, $k);

// This code is contributed by m_kit
?>
```

Output:

```
6
```

Improved By : jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/turn-particular-bit-number-2/

Chapter 188

Increment a number without using ++ or +

Increment a number without using ++ or + - GeeksforGeeks

The task is to Increment a number without using ++ and + operators.

Examples:

```
Input : 3
Output : 4

Input : 9
Output : 10
```

The idea is based on the fact that the negative numbers are stored using 2's complement form. 2's complement form is obtained by inverting bits and then adding one. So if we invert all bits of given number and apply negative sign, we get the number plus 1.

C++

```
 // CPP program to increment an unsigned
// int using bitwise operators.
#include <bits/stdc++.h>
using namespace std;

// function that increment the value.
int increment(unsigned int i)
{
    // Invert bits and apply negative sign
    i = -(~i);
```

```
    return i;
}

// Driver code
int main()
{
    int n = 3;
    cout << increment(n);
    return 0;
}
```

Java

```
 // Java program to increment
// an unsigned int using
// bitwise operators.
import java.io.*;

class GFG
{

// function that increment
// the value.
static long increment(long i)
{
    // Invert bits and
    // apply negative sign
    i = -(~i);

    return i;
}

// Driver code
public static void main (String[] args)
{
    long n = 3;
    System.out.print( increment(n));
}
}

// This code is contributed
// by inder_verma.
```

Python3

```
 # Python3 program to increment
```

```
# an unsigned int using
# bitwise operators.

# function that increment the value.
def increment(i):
    # Invert bits and
    # apply negative sign
    i = -(~i);

    return i;

# Driver code
if __name__ == "__main__":
    n = 3;
    print(increment(n));

# This code is contributed by mits
```

C#

```
 // C# program to increment
// an unsigned int using
// bitwise operators.
using System;

class GFG
{

// function that increment
// the value.
static long increment(long i)
{
    // Invert bits and
    // apply negative sign
    i = -(~i);

    return i;
}

// Driver code
public static void Main ()
{
    long n = 3;
    Console.WriteLine(increment(n));
}
}

// This code is contributed
```

```
// by inder_verma.
```

PHP

```
 <?php
// PHP program to increment
// an unsigned int using
// bitwise operators.

// function that increment the value.
function increment($i)
{
    // Invert bits and
    // apply negative sign
    $i = -(~$i);

    return $i;
}

// Driver code
$n = 3;
echo increment($n);

// This code is contributed by mits
?>
```

Output:

```
4
```

It also works for characters.
Example:

```
Input : a
Output : b

Input : o
Output : p
```

C++

```
 // CPP program to increment an unsigned
// char using bitwise operators.
#include <bits/stdc++.h>
```

```
using namespace std;

// function that increment the value.
char increment(unsigned char i)
{

    // Invert bits and apply negative sign
    i = -(~i);
    return i;
}

// Driver code
int main()
{
    char n = 'a';
    cout << increment(n);
    return 0;
}
```

Java

```
 // Java program to increment
// an unsigned char using
// bitwise operators.
class GFG
{
// function that increment the value.
static char increment(char i)
{

    // Invert bits and apply
    // negative sign
    int i1 = -(~(int)(i));
    return (char)(i1);
}

// Driver code
public static void main(String[] args)
{
    char n = 'a';
    System.out.println(increment(n));
}
}

// This code is contributed by mits
```

Python3

```
 # Python3 program to increment
# an unsigned char using
# bitwise operators.

# function that increment
# the value.
def increment(i):

    # Invert bits and
    # apply negative sign
    i = -(~ord(i));
    return chr(i);

# Driver code
n = 'a';
print(increment(n));

# This code is contributed by mits
```

C#

```
// C# program to increment
// an unsigned char using
// bitwise operators.
class GFG
{
// function that increment the value.
static char increment(char i)
{
// Invert bits and apply
// negative sign
int i1 = -(~(int)(i));
return (char)(i1);
}
// Driver code
static void Main()
{
char n = 'a';
System.Console.WriteLine(increment(n));
}
}
// This code is contributed by mits
```

PHP

```
 <?php
// PHP program to increment
```

```
// an unsigned char using
// bitwise operators.

// function that increment
// the value.
function increment($i)
{

    // Invert bits and
    // apply negative sign
    $i = -(~ord($i));
    return chr($i);
}

// Driver code
$n = 'a';
echo increment($n);

// This code is contributed by mits
?>
```

Output:

```
b
```

Improved By : inderDuMCA, Mithun Kumar

Source

https://www.geeksforgeeks.org/increment-a-number-without-using-or/

Chapter 189

Increment a number by one by manipulating the bits

Increment a number by one by manipulating the bits - GeeksforGeeks

Given a non-negative integer **n**. The problem is to increment **n** by 1 by manipulating the bits of **n**.

Examples :

```
Input : 6
Output : 7

Input : 15
Output : 16
```

Approach: Following are the steps:

1. Get the position of rightmost unset bit of **n**. Let this position be **k**.
2. Set the k-th bit of n.
3. Toggle the last k-1 bits of **n**.
4. Finally, return **n**.

C++

```
 // C++ implementation to increment a number
// by one by manipulating the bits
#include <bits/stdc++.h>
using namespace std;

// function to find the position
```

```
// of rightmost set bit
int getPosOfRightmostSetBit(int n)
{
    return log2(n & -n);
}

// function to toggle the last m bits
unsigned int toggleLastKBits(unsigned int n,
                             unsigned int k)
{
    // calculating a number 'num' having 'm' bits
    // and all are set
    unsigned int num = (1 << k) - 1;

    // toggle the last m bits and return the number
    return (n ^ num);
}

// function to increment a number by one
// by manipulating the bits
unsigned int incrementByOne(unsigned int n)
{
    // get position of rightmost unset bit
    // if all bits of 'n' are set, then the
    // bit left to the MSB is the rightmost
    // unset bit
    int k = getPosOfRightmostSetBit(~n);

    // kth bit of n is being set by this operation
    n = ((1 << k) | n);

    // from the right toggle all the bits before the
    // k-th bit
    if (k != 0)
        n = toggleLastKBits(n, k);

    // required number
    return n;
}

// Driver program to test above
int main()
{
    unsigned int n = 15;
    cout << incrementByOne(n);
    return 0;
}
```

Java

```
 // Java implementation to increment a number
// by one by manipulating the bits
import java.io.*;
import java.util.*;

class GFG {

    // function to find the position
    // of rightmost set bit
    static int getPosOfRightmostSetBit(int n)
    {
        return (int)(Math.log(n & -n) / Math.log(2));
    }

    // function to toggle the last m bits
    static int toggleLastKBits( int n, int k)
    {
        // calculating a number 'num' having
        // 'm' bits and all are set
        int num = (1 << k) - 1;

        // toggle the last m bits and return
        // the number
        return (n ^ num);
    }

    // function to increment a number by one
    // by manipulating the bits
    static int incrementByOne( int n)
    {
        // get position of rightmost unset bit
        // if all bits of 'n' are set, then the
        // bit left to the MSB is the rightmost
        // unset bit
        int k = getPosOfRightmostSetBit(~n);

        // kth bit of n is being set
        // by this operation
        n = ((1 << k) | n);

        // from the right toggle all
        // the bits before the k-th bit
        if (k != 0)
            n = toggleLastKBits(n, k);

        // required number
```

```
        return n;

    }

    // Driver Program
    public static void main (String[] args)
    {
        int n = 15;
        System.out.println(incrementByOne(n));

    }
}

// This code is contributed by Gitanjali.
```

Python 3

```
 # python 3 implementation
# to increment a number
# by one by manipulating
# the bits
import math

# function to find the
# position of rightmost
# set bit
def getPosOfRightmostSetBit(n) :
    return math.log2(n & -n)


# function to toggle the last m bits
def toggleLastKBits(n, k) :
    # calculating a number
    # 'num' having 'm' bits
    # and all are set
    num = (1 << (int)(k)) - 1

    # toggle the last m bits and
    # return the number
    return (n ^ num)


# function to increment
# a number by one by
# manipulating the bits
def incrementByOne(n) :

    # get position of rightmost
```

```
    # unset bit if all bits of
    # 'n' are set, then the bit
    # left to the MSB is the
    # rightmost unset bit
    k = getPosOfRightmostSetBit(~n)

    # kth bit of n is being
    # set by this operation
    n = ((1 << (int)(k)) | n)

    # from the right toggle
    # all the bits before the
    # k-th bit
    if (k != 0) :
        n = toggleLastKBits(n, k)

    # required number
    return n


# Driver program
n = 15
print(incrementByOne(n))


# This code is contributed
# by Nikita Tiwari.
```

C#

```
 // C# implementation to increment a number
// by one by manipulating the bits
using System;

class GFG {

    // function to find the position
    // of rightmost set bit
    static int getPosOfRightmostSetBit(int n)
    {
        return (int)(Math.Log(n & -n) / Math.Log(2));
    }

    // function to toggle the last m bits
    static int toggleLastKBits( int n, int k)
    {

        // calculating a number 'num' having
```

```
        // 'm' bits and all are set
        int num = (1 << k) - 1;

        // toggle the last m bits and return
        // the number
        return (n ^ num);
    }

    // function to increment a number by one
    // by manipulating the bits
    static int incrementByOne( int n)
    {

        // get position of rightmost unset bit
        // if all bits of 'n' are set, then the
        // bit left to the MSB is the rightmost
        // unset bit
        int k = getPosOfRightmostSetBit(~n);

        // kth bit of n is being set
        // by this operation
        n = ((1 << k) | n);

        // from the right toggle all
        // the bits before the k-th bit
        if (k != 0)
            n = toggleLastKBits(n, k);

        // required number
        return n;

    }

    // Driver Program
    public static void Main ()
    {
        int n = 15;

        Console.WriteLine(incrementByOne(n));

    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
```

```
// PHP implementation to increment a number
// by one by manipulating the bits

// function to find the position
// of rightmost set bit
function getPosOfRightmostSetBit($n)
{
    $t = $n & -$n;
    return log($t, 2);
}

// function to toggle the last m bits
function toggleLastKBits($n, $k)
{
    // calculating a number 'num'
    // having 'm' bits and all are set
    $num = (1 << $k) - 1;

    // toggle the last m bits and
    // return the number
    return ($n ^ $num);
}

// function to increment a number by one
// by manipulating the bits
function incrementByOne($n)
{
    // get position of rightmost unset bit
    // if all bits of 'n' are set, then the
    // bit left to the MSB is the rightmost
    // unset bit
    $k = getPosOfRightmostSetBit(~$n);

    // kth bit of n is being set
    // by this operation
    $n = ((1 << $k) | $n);

    // from the right toggle all the
    // bits before the k-th
    if ($k != 0)
        $n = toggleLastKBits($n, $k);

    // required number
    return $n;
}

// Driver code
$n = 15;
```

```
echo incrementByOne($n);

// This code is contributed by Mithun Kumar
?>
```

Output :

```
16
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/increment-number-one-manipulating-bits/

Chapter 190

Inserting M into N such that m starts at bit j and ends at bit i | Set-2

Inserting M into N such that m starts at bit j and ends at bit i | Set-2 - GeeksforGeeks

Given two 32-bit numbers, N and M, and two-bit positions, i and j. Write a method to insert M into N such that M starts at bit j and ends at bit i. You can assume that the bits j through i have enough space to fit all of M. Assuming index start from 0.

Examples:

```
a)  N = 1024 (10000000000),
    M = 19 (10011),
    i = 2, j = 6
    Output : 1100 (10001001100)
b)  N = 1201 (10010110001)
    M = 8 (1000)
    i = 3, j = 6
    Output: 1217 (10011000001)
```

This problem has been already discussed in the previous post. In this post, a different approach is discussed.
Approach: The idea is very straightforward, following is the stepwise procedure –

- Capture all the bits before i in N, that is from i-1 to 0.
- We don't want to alter those bits so we will capture them and use later
- Clear all bits from j to 0
- Insert M into N at position j to i
- Insert captured bits at there respective position ie. from i-1 to 0

Let's try to solve example b for a clear explanation of the procedure.

Capturing bits i-1 to 0 in N

Create a mask whose i-1 to 0 bits are 1 and rest are 0. Shift 1 to i position in left and subtract 1 from this to get a bit mask having i-1 to 0 bits set.

```
capture_mask = ( 1 << i ) - 1
```

So for example b, mask will be –

```
capture_mask = ( 1 << 3 ) - 1
Now capture_mask = 1(001)
```

Now *AND* this mask with N, i-1 to 0 bits will remain same while rest bits become 0. Thus we are left with i-1 to 0 bits only.

```
captured_bits = N & capture_mask
```

for example b, captured bits will be –

```
captured_bits = N & capture_mask
Now capture_mask = 1 (001)
```

Clearing bits from j to 0 in N

Since bits have been captured, clear the bits j to 0 without loosing bits i-1 to 0. Create a mask whose j to 0 bits are 0 while the rest are 1. Create such mask by shifting -1 to j+1 position in left.

```
clear_mask = -1 << ( j + 1 )
```

Now to clear bits j to 0, *AND* mask with N.

```
N  &= clear_mask
```

For example b will be:

```
N &= clear_mask
Now N = 1152 (10010000000)
```

Inserting M in N

Now because N has bits from j to 0 cleared, just fit in M into N and shift M i position to left to align the MSB of M with position j in N.

```
M <<= i
```

For example b-

```
M <<= 3
Now M = 8 << 3 = 64 (1000000)
```

Do a *OR* of M with N to insert M at desired position.

```
N |= M
```

For example b –

```
N |= M
Now N = 1152 | 64 = 1216 (10011000000)
```

Inserting captured bits into N

Till now M has been inserted into N. Now the only thing left is merging back the captured bits at position i-1 to 0. This can be simply done by OR of N and captured bits –

```
N |= captured_bits
```

For example b –

```
N |= captured_bits
N = 1216 | 1 = 1217 (10011000001)
```

So finally after insertion, the below bitset is obtained.

```
N(before) = 1201 (10010110001)
N(after) = 1201 (10011000001)
```

Below is the implementation of the above approach:

C++

```
 // C++ program to insert 32-bit number
// M into N using bit magic
#include <bits/stdc++.h>
using namespace std;

// print binary representation of n
void bin(unsigned n)
{
    if (n > 1)
        bin(n / 2);
    printf("%d", n % 2);
}

// Insert m into n
int insertion(int n, int m, int i, int j)
{
    int clear_mask = -1 << (j + 1);
    int capture_mask = (1 << i) - 1;

    // Capturing bits from i-1 to 0
    int captured_bits = n & capture_mask;

    // Clearing bits from j to 0
    n &= clear_mask;

    // Shiftng m to align with n
    m = m << i;

    // Insert m into n
    n |= m;

    // Insert captured bits
    n |= captured_bits;

    return n;
}

// Driver Code
int main()
{

    // print original bitset
    int N = 1201, M = 8, i = 3, j = 6;
    cout << "N = " << N << "(";
    bin(N);
```

```
    cout << ")"
         << "\n";

    // print original bitset
    cout << "M = " << M << "(";
    bin(M);
    cout << ")"
         << "\n";

    // Call function to insert M to N
    N = insertion(N, M, i, j);
    cout << "After inserting M into N from 3 to 6"
         << "\n";

    // Print the inserted bitset
    cout << "N = " << N << "(";
    bin(N);
    cout << ")"
         << "\n";
    return 0;
}
```

Java

```
 // Java program to insert
// 32-bit number M into N
// using bit magic
import java.io.*;

class GFG
{

// print binary
// representation of n
static void bin(long n)
{
    if (n > 1)
        bin(n / 2);
    System.out.print(n % 2);
}

// Insert m into n
static int insertion(int n, int m,
                     int i, int j)
{
    int clear_mask = -1 << (j + 1);
    int capture_mask = (1 << i) - 1;
```

```
    // Capturing bits from i-1 to 0
    int captured_bits = n & capture_mask;

    // Clearing bits from j to 0
    n &= clear_mask;

    // Shiftng m to align with n
    m = m << i;

    // Insert m into n
    n |= m;

    // Insert captured bits
    n |= captured_bits;

    return n;
}

// Driver Code
public static void main (String[] args)
{
    // print original bitset
    int N = 1201, M = 8, i = 3, j = 6;
    System.out.print("N = " + N + "(");
    bin(N);
    System.out.println(")");

    // print original bitset
    System.out.print("M = " + M + "(");
    bin(M);
    System.out.println(")");

    // Call function to insert M to N
    N = insertion(N, M, i, j);
    System.out.println( "After inserting M " +
                        "into N from 3 to 6");

    // Print the inserted bitset
    System.out.print("N = " + N + "(");
    bin(N);
    System.out.println(")");
}
}

// This code is contributed
// by inder_verma.
```

C#

```
 // C# program to insert
// 32-bit number M into N
// using bit magic
using System;

class GFG
{

// print binary
// representation of n
static void bin(long n)
{
    if (n > 1)
        bin(n / 2);
    Console.Write(n % 2);
}

// Insert m into n
static int insertion(int n, int m,
                     int i, int j)
{
    int clear_mask = -1 << (j + 1);
    int capture_mask = (1 << i) - 1;

    // Capturing bits from i-1 to 0
    int captured_bits = n & capture_mask;

    // Clearing bits from j to 0
    n &= clear_mask;

    // Shiftng m to align with n
    m = m << i;

    // Insert m into n
    n |= m;

    // Insert captured bits
    n |= captured_bits;

    return n;
}

// Driver Code
static public void Main (String []args)
{
    // print original bitset
    int N = 1201, M = 8, i = 3, j = 6;
    Console.Write("N = " + N + "(");
```

```
    bin(N);
    Console.WriteLine(")");

    // print original bitset
    Console.Write("M = " + M + "(");
    bin(M);
    Console.WriteLine(")");

    // Call function to
    // insert M to N
    N = insertion(N, M, i, j);
    Console.WriteLine("After inserting M " +
                      "into N from 3 to 6");

    // Print the inserted bitset
    Console.Write("N = " + N + "(");
    bin(N);
    Console.WriteLine(")");
}
}

// This code is contributed
// by Arnab Kundu
```

Python

```
 # Python 3 program to insert 32-bit number
# M into N using bit magic

# insert M into N
def insertion(n, m, i, j):

    clear_mask = -1 << (j + 1)
    capture_mask = (1 << i) - 1

    # Capturing bits from i-1 to 0
    captured_bits = n & capture_mask

    # Clearing bits from j to 0
    n &= clear_mask

    # Shiftng m to align with n
    m = m << i

    # Insert m into n
    n |= m

    # Insert captured bits
```

```
    n |= captured_bits

    return n

# Driver
def main():
    N = 1201; M = 8; i = 3; j = 6
    print("N = {}({})".format(N, bin(N)))
    print("M = {}({})".format(M, bin(M)))
    N = insertion(N, M, i, j)
    print("***After inserting M into N***")
    print("N = {}({})".format(N, bin(N)))

if __name__ == '__main__':
    main()
```

Output:

```
N = 1201(10010110001)
M = 8(1000)
After inserting M into N from 3 to 6
N = 1217(10011000001)
```

Improved By : inderDuMCA, andrew1234

Source

https://www.geeksforgeeks.org/inserting-m-into-n-such-that-m-starts-at-bit-j-and-ends-at-bit-i-set-2/

Chapter 191

Inserting m into n such that m starts at bit j and ends at bit i.

Inserting m into n such that m starts at bit j and ends at bit i. - GeeksforGeeks

We are given two numbers n and m, and two-bit positions, i and j. Insert bits of m into n starting from j to i. We can assume that the bits j through i have enough space to fit all of m. That is, if m = 10011, you can assume that there are at least 5 bits between j and i. You would not, for example, have j = 3 and i = 2, because m could not fully fit between bit 3 and bit 2.

Examples :

```
Input : n = 1024
        m = 19
        i = 2
        j = 6;
Output : n = 1100
Binary representations of input numbers
m in binary is (10011)2
n in binary is (10000000000)2
Binary representations of output number
(10000000000)2

Input : n = 5
        m = 3
        i = 1
        j = 2
Output : 7
```

Algorithm :

```
1. Clear the bits j through i in n

2. Shift m so that it lines up with bits j through i

3. Return Bitwise AND of m and n.
```

The trickiest part is Step 1. How do we clear the bits in n? We can do this with a mask. This mask will have all 1s, except for 0s in the bits j through i. We create this mask by creating the left half of the mask first, and then the right half.

Following is the implementation of the above approach.

C++

```
 // C++ program for implementation of updateBits()
#include <bits/stdc++.h>
using namespace std;

// Function to updateBits M insert to N.
int updateBits(int n, int m, int i, int j)
{
    /* Create a mask to clear bits i through j
      in n. EXAMPLE: i = 2, j = 4. Result
      should be 11100011. For simplicity, we'll
      use just 8 bits for the example. */

    int allOnes = ~0; // will equal sequence of all ls

    // ls before position j, then 0s. left = 11100000
    int left= allOnes << (j + 1);

    // l's after position i. right = 00000011
    int right = ((1 << i) - 1);

    // All ls, except for 0s between i and j. mask 11100011
    int mask = left | right;

    /* Clear bits j through i then put min there */
    int n_cleared = n & mask; // Clear bits j through i.
    int m_shifted = m << i;   // Move m into correct position.

    return (n_cleared | m_shifted); // OR them, and we're done!
}

// Driver Code
int main()
{
    int n = 1024; // in Binary N= 10000000000
```

```
    int m = 19;   // in Binary M= 10011
    int i = 2, j = 6;

    cout << updateBits(n,m,i,j);

    return 0;
}
```

Java

```
 // Java program for implementation of updateBits()

class UpdateBits
{
    // Function to updateBits M insert to N.
    static int updateBits(int n, int m, int i, int j)
    {
        /* Create a mask to clear bits i through j
          in n. EXAMPLE: i = 2, j = 4. Result
          should be 11100011. For simplicity, we'll
          use just 8 bits for the example. */

        int allOnes = ~0; // will equal sequence of all ls

        // ls before position j, then 0s. left = 11100000
        int left= allOnes << (j + 1);

        // l's after position i. right = 00000011
        int right = ((1 << i) - 1);

        // All ls, except for 0s between i and j. mask 11100011
        int mask = left | right;

        /* Clear bits j through i then put min there */
        // Clear bits j through i.
        int n_cleared = n & mask;
        // Move m into correct position.
        int m_shifted = m << i;

        // OR them, and we're done!
        return (n_cleared | m_shifted);
    }

    public static void main (String[] args)
    {
        // in Binary N= 10000000000
        int n = 1024;
```

```
        // in Binary M= 10011
        int m = 19;

        int i = 2, j = 6;

        System.out.println(updateBits(n,m,i,j));
    }
}
```

Python3

```
 # Python3 program for implementation
# of updateBits()

# Function to updateBits M insert to N.
def updateBits(n, m, i, j):

    # Create a mask to clear bits i through
    # j in n. EXAMPLE: i = 2, j = 4. Result
    # should be 11100011. For simplicity,
    # we'll use just 8 bits for the example.

    # will equal sequence of all ls
    allOnes = ~0

    # ls before position j,
    # then 0s. left = 11100000
    left = allOnes << (j + 1)

    # l's after position i. right = 00000011
    right = ((1 << i) - 1)

    # All ls, except for 0s between
    # i and j. mask 11100011
    mask = left | right

    # Clear bits j through i then put min there
    n_cleared = n & mask

    # Move m into correct position.
    m_shifted = m << i

    return (n_cleared | m_shifted)


# Driver Code
n = 1024 # in Binary N = 10000000000
m = 19   # in Binary M = 10011
```

```
i = 2; j = 6
print(updateBits(n, m, i, j))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program for implementation of
// updateBits()
using System;

class GFG {

    // Function to updateBits M
    // insert to N.
    static int updateBits(int n, int m,
                          int i, int j)
    {

        /* Create a mask to clear bits i
          through j in n. EXAMPLE: i = 2,
          j = 4. Result should be 11100011.
          For simplicity, we'll use just 8
          bits for the example. */

        // will equal sequence of all ls
        int allOnes = ~0;

        // ls before position j, then 0s.
        // left = 11100000
        int left= allOnes << (j + 1);

        // l's after position i.
        // right = 00000011
        int right = ((1 << i) - 1);

        // All ls, except for 0s between i
        // and j. mask 11100011
        int mask = left | right;

        /* Clear bits j through i then put
        min there */
        // Clear bits j through i.
        int n_cleared = n & mask;

        // Move m into correct position.
        int m_shifted = m << i;
```

```
        // OR them, and we're done!
        return (n_cleared | m_shifted);
    }

    public static void Main()
    {

        // in Binary N= 10000000000
        int n = 1024;

        // in Binary M= 10011
        int m = 19;
        int i = 2, j = 6;

        Console.WriteLine(updateBits(n, m, i, j));
    }
}

//This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program for implementation
// of updateBits()

// Function to updateBits
// M insert to N.

function updateBits($n, $m, $i, $j)
{
    // Create a mask to clear
    // bits i through j in n.
    // EXAMPLE: i = 2, j = 4.
    // Result should be 11100011.
    // For simplicity, we'll use
    // just 8 bits for the example.

    // will equal sequence of all ls
    $allOnes = ~0;

    // ls before position j, then
    // 0s. left = 11100000
    $left= $allOnes << ($j + 1);

    // l's after position i.
    // right = 00000011
    $right = ((1 << $i) - 1);
```

```
    // All ls, except for 0s between
    // i and j. mask 11100011
    $mask = $left | $right;

    // Clear bits j through i
    // then put min there

    // Clear bits j through i.
    $n_cleared = $n & $mask;

    // Move m into correct position.
    $m_shifted = $m << $i;

    // OR them, and we're done!
    return ($n_cleared | $m_shifted);
}

// Driver Code

// in Binary N= 10000000000
$n = 1024;

// in Binary M= 10011
$m = 19;
$i = 2;
$j = 6;

echo updateBits($n, $m, $i, $j);

// This code is contributed by Ajit
?>
```

Output :

```
 1100  // in Binary (10001001100)2
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/insertion-m-n-m-starts-bit-j-ends-bit/

Chapter 192

Invert actual bits of a number

Invert actual bits of a number - GeeksforGeeks

Given a non-negative integer n. The problem is to invert the bits of n and print the number obtained after inverting the bits. Note that the actual binary representation of the number is being considered for inverting the bits, no leading 0's are being considered.
Examples :

```
Input : 11
Output : 4
(11)10 = (1011)[2]
After inverting the bits, we get:
(0100)2 = (4)10.

Input : 10
Output : 5
(10)10 = (1010)2.
After reversing the bits we get:
(0101)2 = (101)2
        = (5)10.
```

Method 1 (Using bitwise operators)

Prerequisite :Toggling k-th bit of a number

C++

```
 // CPP program to invert actual bits
// of a number.
#include <bits/stdc++.h>
using namespace std;
```

```
void invertBits(int num)
{
    // calculating number of bits
    // in the number
    int x = log2(num) + 1;

    // Inverting the bits one by one
    for (int i = 0; i < x; i++)
       num = (num ^ (1 << i));

    cout << num;
}

// Driver code
int main()
{
    int num = 11;
    invertBits(num);
    return 0;
}
```

Java

```
 // Java program to invert
// actual bits of a number.
import java.io.*;

class GFG
{
    static void invertBits(int num)
    {
        // calculating number of
        // bits in the number
        int x = (int)(Math.log(num) /
                      Math.log(2)) + 1;

        // Inverting the
        // bits one by one
        for (int i = 0; i < x; i++)
        num = (num ^ (1 << i));

        System.out.println(num);
    }

    // Driver code
    public static void main (String[] args)
    {
        int num = 11;
```

```
        invertBits(num);
    }

}

// This code is contributed
// by Anuj_67
```

C#

```
 // C# program to invert
// actual bits of a number.
using System;

class GFG
{
    static void invertBits(int num)
    {
        // calculating number of
        // bits in the number
        int x = (int)(Math.Log(num) /
                       Math.Log(2)) + 1;

        // Inverting the
        // bits one by one
        for (int i = 0; i < x; i++)
        num = (num ^ (1 << i));

        Console.WriteLine(num);
    }

    // Driver code
    public static void Main ()
    {
        int num = 11;
        invertBits(num);
    }

}

// This code is contributed
// by Anuj_67
```

PHP

```
 <?php
// PHP program to invert actual bits
```

```
// of a number.

function invertBits( $num)
{

    // calculating number of bits
    // in the number
    $x = log($num) + 1;

    // Inverting the bits one by one
    for($i = 0; $i < $x; $i++)
    $num = ($num ^ (1 << $i));

    echo $num;
}

    // Driver code
    $num = 11;
    invertBits($num);

// This code is contributed by anuj_67.
?>
```

Output:

```
4
```

Time complexity : O(log n)
Auxiliary space : O(1)

Method 2 (Using Bitset)
Here we use the flip() of bitset to invert the bits of the number, in order to avoid flipping the leading zeroes in the binary representation of number, we have calculated the number of bits in the binary representation, and flipped only the actual bits of number.We have used to_ulong() to convert bitset to number.

C++

```
 // CPP program to invert actual bits
// of a number.
#include <bits/stdc++.h>
using namespace std;

void invertBits(int num)
{
    // calculating number of bits
    // in the number
```

```
    int x = log2(num) + 1;

    // Considering number to be 32 bit integer;
    bitset<32> b(num);

    // reversing the bits one by one
    for (int i = 0; i < x; i++)
        b.flip(i);

    // converting bitset to number
    cout << b.to_ulong();
}

// Driver code
int main()
{
    int num = 11;
    invertBits(num);
    return 0;
}
```

C#

```
 // C# program to invert actual
// bits of a number.
using System;

class GFG
{
    static void invertBits(int num)
    {
        // calculating number of
        // bits in the number
        int x = (int)Math.Log(num, 2) + 1;

        // Inverting the bits
        // one by one
        for (int i = 0; i < x; i++)
        num = (num ^ (1 << i));

        Console.Write(num);
    }

    // Driver code
    static void Main()
    {
        int num = 11;
        invertBits(num);
```

```
    }

}

// This code is contributed by Anuj_67
```

Output :

```
4
```

Time complexity : O(log n)
Auxiliary space : O(1)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/invert-actual-bits-number/

Chapter 193

Josephus Problem Using Bit Magic

Josephus Problem Using Bit Magic - GeeksforGeeks

The Problem

This problem is named after Flavius Josephus a Jewish historian who fought against the Romans. According to Josephus him and his group of Jewish soldiers were cornered & surrounded by the Romans inside a cave and they chose murder and suicide inside of surrender and capture. They decided that all the soldiers will sit in a circle and starting from the soldier sitting at the first position every soldier will kill the soldier to their in a sequential way. So if there are 5 soldiers sitting in a circle with positions numbered as 1, 2, 3, 4, 5. The soldier 1 kills 2, then 3 kills 4, then 5 kills 1, then 3 kills 5 and since 3 is the only one left then 3 commits suicide.

Now Josephus doesn't want to get murdered or commit suicide. He would rather be captured by the Romans and is presented with a problem. He has to figure out at which position should he sit in a circle (provided there are n men in total and the man sitting at position 1 gets the first chance to murder) so that he is the last man standing and instead of committing suicide he will surrender to the Romans.

The Pattern

If you work this out for different values of n the you will find a pattern here. If n is a true power of 2 then the answer is always 1. For every n greater than that power of 2 the answer is incremented by 2.

n soldiers	$2^a + l$	Survivor W(n) = 2l + 1
1	1 + 0	2 * 0 + 1 = 1
2	2 + 0	2 * 0 + 1 = 1
3	2 + 1	2 * 1 + 1 = 3
4	4 + 0	2 * 0 + 1 = 1
5	4 + 1	2 * 1 + 1 = 3

n soldiers	$2^a + l$	Survivor W(n) = 2l + 1
6	4 + 2	2 * 2 + 1 = 5
7	4 + 3	2 * 3 + 1 = 7
8	8 + 0	2 * 0 + 1 = 1
9	8 + 1	2 * 1 + 1 = 3
10	8 + 2	2 * 2 + 1 = 5
11	8 + 3	2 * 3 + 1 = 7
12	8 + 4	2 * 4 + 1 = 9

Now for every n the right position for Josephus can be found out by deducting the biggest possible power of 2 from the number and we get the answer (provided that value of n is not a pure power of 2 otherwise the answer is 1)

$N = 2^a$ + something

Where, a = biggest possible power

The Trick

Whenever someone talks about the powers of 2 the first word that comes to mind is "binary". The solution to this problem is much is easier and shorter in binary than in decimal. There is a trick to this. Since we need to deduct the biggest possible power of in binary that number is the Most Significant Bit. In the original Josephus problem there were 40 other soldiers along with Josephus which makes **n = 41**. 41 in binary is 101001. If we shift the MSB i.e. the leftmost 1 to the rightmost place we get 010011 which is 19 (in decimal) which is the answer. This is true for all cases. This can be done easily using bit manipulation.

C

```
 // C program for josephus problem

#include <stdio.h>

// function to find the position of the Most
// Significant Bit
int msbPos(int n)
{
    int pos = 0;
    while (n != 0) {
        pos++;

        // keeps shifting bits to the right
        // until we are left with 0
        n = n >> 1;
    }
    return pos;
}

// function to return at which place Josephus
```

```
// should sit to avoid being killed
int josephify(int n)
{
    /*  Getting the position of the Most Significant
        Bit(MSB). The leftmost '1'. If the number is
       '41' then its binary is '101001'.
        So msbPos(41) = 6   */
    int position = msbPos(n);

    /* 'j' stores the number with which to XOR the
        number 'n'. Since we need '100000'
       We will do 1<<6-1 to get '100000' */
    int j = 1 << (position - 1);

    /* Toggling the Most Significant Bit. Changing
       the leftmost '1' to '0'.
       101001 ^ 100000 = 001001 (9) */
    n = n ^ j;

    /*  Left-shifting once to add an extra '0' to
        the right end of the binary number
        001001 = 010010 (18) */
    n = n << 1;

    /* Toggling the '0' at the end to '1' which
       is essentially the same as putting the
       MSB at the rightmost place. 010010 | 1
       = 010011 (19) */
    n = n | 1;

    return n;
}

// hard coded driver main function to run the program
int main()
{
    int n = 41;
    printf("%d\n", josephify(n));
    return 0;
}
```

Java

```
 // Java program for josephus problem

public class GFG
{
    // method to find the position of the Most
```

```
// Significant Bit
static int msbPos(int n)
{
    int pos = 0;
    while (n != 0) {
        pos++;

        // keeps shifting bits to the right
        // until we are left with 0
        n = n >> 1;
    }
    return pos;
}

// method to return at which place Josephus
// should sit to avoid being killed
static int josephify(int n)
{
    /*  Getting the position of the Most Significant
        Bit(MSB). The leftmost '1'. If the number is
       '41' then its binary is '101001'.
        So msbPos(41) = 6   */
    int position = msbPos(n);

    /* 'j' stores the number with which to XOR the
        number 'n'. Since we need '100000'
       We will do 1<<6-1 to get '100000' */
    int j = 1 << (position - 1);

    /* Toggling the Most Significant Bit. Changing
       the leftmost '1' to '0'.
       101001 ^ 100000 = 001001 (9) */
    n = n ^ j;

    /*  Left-shifting once to add an extra '0' to
        the right end of the binary number
        001001 = 010010 (18) */
    n = n << 1;

    /* Toggling the '0' at the end to '1' which
       is essentially the same as putting the
       MSB at the rightmost place. 010010 | 1
       = 010011 (19) */
    n = n | 1;

    return n;
}
```

```
    // Driver Method
    public static void main(String[] args)
    {
        int n = 41;
        System.out.println(josephify(n));
    }
}
```

Python3

```
 # Python3 program for josephus problem

# Function to find the position
# of the Most Significant Bit
def msbPos(n):
    pos = 0
    while n != 0:
        pos += 1
        n = n >> 1
    return pos

# Function to return at which
# place Josephus should sit to
# avoid being killed
def josephify(n):

    # Getting the position of the Most
    # Significant Bit(MSB). The leftmost '1'.
    # If the number is '41' then its binary
    # is '101001'. So msbPos(41) = 6
    position = msbPos(n)

    # 'j' stores the number with which to XOR
    # the number 'n'. Since we need '100000'
    # We will do 1<<6-1 to get '100000'
    j = 1 << (position - 1)

    # Toggling the Most Significant Bit.
    # Changing the leftmost '1' to '0'.
    # 101001 ^ 100000 = 001001 (9)
    n = n ^ j

    # Left-shifting once to add an extra '0'
    # to the right end of the binary number
    # 001001 = 010010 (18)
    n = n << 1

    # Toggling the '0' at the end to '1'
```

```
    # which is essentially the same as
    # putting the MSB at the rightmost
    # place. 010010 | 1 = 010011 (19)
    n = n | 1

    return n

# Driver Code
n = 41
print (josephify(n))

# This code is contributed by Shreyanshi Arun.
```

C#

```
 // C# program for Josephus Problem
using System;

public class GFG
{

    // Method to find the position
    // of the Most Significant Bit
    static int msbPos(int n)
    {
        int pos = 0;
        while (n != 0) {
            pos++;

            // keeps shifting bits to the right
            // until we are left with 0
            n = n >> 1;
        }
        return pos;
    }

    // method to return at which place Josephus
    // should sit to avoid being killed
    static int josephify(int n)
    {

        // Getting the position of the Most Significant
        // Bit(MSB). The leftmost '1'. If the number is
        // '41' then its binary is '101001'.
        // So msbPos(41) = 6
        int position = msbPos(n);

        // 'j' stores the number with which to XOR
```

```
        // the number 'n'. Since we need '100000'
        // We will do 1<<6-1 to get '100000'
        int j = 1 << (position - 1);

        // Toggling the Most Significant Bit.
        // Changing the leftmost '1' to '0'.
        // 101001 ^ 100000 = 001001 (9)
        n = n ^ j;

        // Left-shifting once to add an extra '0'
        // to the right end of the binary number
        // 001001 = 010010 (18)
        n = n << 1;

        // Toggling the '0' at the end to '1' which
        // is essentially the same as putting the
        // MSB at the rightmost place. 010010 | 1
        // = 010011 (19)
        n = n | 1;

        return n;
    }

    // Driver code
    public static void Main()
    {
        int n = 41;
        Console.WriteLine(josephify(n));
    }
}

// This code is contributed by vt_m .
```

Output:

```
19
```

References:

1. Numberphile
2. Wikipedia

Previous articles on the same topic:

1. Josephus problem | Set 1 (A O(n) Solution)
2. Josephus problem | Set 2 (A Simple Solution when k = 2)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/josephus-problem-using-bit-magic/

Chapter 194

Josephus problem | Set 1 (A O(n) Solution)

Josephus problem | Set 1 (A O(n) Solution) - GeeksforGeeks

In computer science and mathematics, the Josephus Problem (or Josephus permutation) is a theoretical problem. Following is the problem statement:

There are n people standing in a circle waiting to be executed. The counting out begins at some point in the circle and proceeds around the circle in a fixed direction. In each step, a certain number of people are skipped and the next person is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom. Given the total number of persons n and a number k which indicates that k-1 persons are skipped and kth person is killed in circle. The task is to choose the place in the initial circle so that you are the last one remaining and so survive.

For example, if n = 5 and k = 2, then the safe position is 3. Firstly, the person at position 2 is killed, then person at position 4 is killed, then person at position 1 is killed. Finally, the person at position 5 is killed. So the person at position 3 survives.
If n = 7 and k = 3, then the safe position is 4. The persons at positions 3, 6, 2, 7, 5, 1 are killed in order, and person at position 4 survives.

The problem has following recursive structure.

```
josephus(n, k) = (josephus(n - 1, k) + k-1) % n + 1
josephus(1, k) = 1
```

After the first person (kth from begining) is killed, n-1 persons are left. So we call josephus(n – 1, k) to get the position with n-1 persons. But the position returned by josephus(n – 1, k) will consider the position starting from k%n + 1. So, we must make adjustments to the position returned by josephus(n – 1, k).

Following is simple recursive implementation of the Josephus problem. The implementation simply follows the recursive structure mentioned above.

C

```
 #include <stdio.h>

int josephus(int n, int k)
{
  if (n == 1)
    return 1;
  else
    /* The position returned by josephus(n - 1, k) is adjusted because the
       recursive call josephus(n - 1, k) considers the original position
       k%n + 1 as position 1 */
    return (josephus(n - 1, k) + k-1) % n + 1;
}

// Driver Program to test above function
int main()
{
  int n = 14;
  int k = 2;
  printf("The chosen place is %d", josephus(n, k));
  return 0;
}
```

Java

```
 // Java code for Josephus Problem
import java.io.*;

class GFG {

static int josephus(int n, int k)
{
if (n == 1)
    return 1;
else
    /* The position returned by josephus(n - 1, k)
    is adjusted because the recursive call
    josephus(n - 1, k) considers the original
    position k%n + 1 as position 1 */
    return (josephus(n - 1, k) + k-1) % n + 1;
}

// Driver Program to test above function
```

```
public static void main(String[] args)
{
int n = 14;
int k = 2;
System.out.println("The chosen place is " + josephus(n, k));
}
}

// This code is contributed by Prerna Saini
```

Python3

```
 # Python code for Josephus Problem

def josephus(n, k):

      if (n == 1):
          return 1
      else:


          # The position returned by
          # josephus(n - 1, k) is adjusted
          # because the recursive call
          # josephus(n - 1, k) considers
          # the original position
          # k%n + 1 as position 1
          return (josephus(n - 1, k) + k-1) % n + 1

# Driver Program to test above function

n = 14
k = 2

print("The chosen place is ", josephus(n, k))

# This code is contributed by
# Sumit Sadhakar
```

C#

```
 // C# code for Josephus Problem
using System;

class GFG {

    static int josephus(int n, int k)
```

```
    {
        if (n == 1)
            return 1;
        else
            /* The position returned
            by josephus(n - 1, k) is
            adjusted because the
            recursive call josephus(n
            - 1, k) considers the
            original position k%n + 1
            as position 1 */
            return (josephus(n - 1, k)
                        + k-1) % n + 1;
    }

    // Driver Program to test above
    // function
    public static void Main()
    {
        int n = 14;
        int k = 2;
        Console.WriteLine("The chosen "
        + "place is " + josephus(n, k));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP code for
// Josephus Problem

function josephus($n, $k)
{
    if ($n == 1)
        return 1;
    else
        /* The position returned by
           josephus(n - 1, k) is
           adjusted because the
           recursive call josephus
           (n - 1, k) considers the
           original position k%n + 1
           as position 1 */
        return (josephus($n - 1, $k) +
                    $k - 1) % $n + 1;
```

```
}

    // Driver Code
    $n = 14;
    $k = 2;
    echo "The chosen place is ", josephus($n, $k);

// This code is contributed by ajit.
?>
```

Output:

```
The chosen place is 13
```

Time Complexity: O(n)

Josephus problem | Set 2 (A Simple Solution when k = 2)

Source:
http://en.wikipedia.org/wiki/Josephus_problem

Improved By : jit_t, vt_m

Source

https://www.geeksforgeeks.org/josephus-problem-set-1-a-on-solution/

Chapter 195

Largest number with binary representation is m 1's and m-1 0's

Largest number with binary representation is m 1's and m-1 0's - GeeksforGeeks

Given n, find the greatest number which is strictly not more then n and whose binary representation consists of m consecutive ones, then m-1 consecutive zeros and nothing else

Examples:

```
Input : n = 7
Output : 6
Explanation: 6's binary representation is 110,
and 7's is 111, so 6 consists of 2 consecutive
1's and then 1 consecutive 0.

Input : 130
Output : 120
Explanation: 28 and 120 are the only numbers <=120,
28 is 11100 consists of 3 consecutive 1's and then
2 consecutive 0's. 120 is 1111000 consists of 4
consecutive 1's and then 3 consecutive 0's. So 120
is the greatest of number<=120 which meets the
given condition.
```

A **naive approach** will be to traverse from 1 to N and check for every binary representation which consists of m consecutive 1's and m-1 consecutive 0's and store the largest of them which meets the given condition.

An **efficient approach** is to observe a pattern of numbers,

[1(1), 6(110), 28(11100), 120(1111000), 496(111110000), ...]

To get the formula for the numbers which satisfies the conditions we take 120 as an example- 120 is represented as 1111000 which has m = 4 1's and m = 3 0's. Converting 1111000 to decimal we get:
2^3+2^4+2^5+2^6 which can be represented as (2^m-1 + 2^m+ 2^m+1 + … 2^m+2, 2^2*m)
2^3*(1+2+2^2+2^3) which can be represented as (2^(m-1)*(1+2+2^2+2^3+..2^(m-1))
2^3*(2^4-1) which can be represented as **[2^(m-1) * (2^m -1)].**
So all the numbers that meet the given condition can be represented as

[2^(m-1) * (2^m -1)]

We can iterate till the number does not exceeds N and print the largest of all possible elements. A closer observation will shows that at **m = 33 it will exceed the 10^18 mark** , so we are calculating the number in unit's time as **log(32)** is near to constant which is required in calculating the **pow** .
So, the overall complexity will be O(1).

C++

```
 // CPP program to find largest number
// smaller than equal to n with m set
// bits then m-1 0 bits.
#include <bits/stdc++.h>
using namespace std;

// Returns largest number with m set
// bits then m-1 0 bits.
long long answer(long long n)
{
    // Start with 2 bits.
    long m = 2;

    // initial answer is 1
    // which meets the given condition
    long long ans = 1;
    long long r = 1;

    // check for all numbers
    while (r < n) {

        // compute the number
        r = (int)(pow(2, m) - 1) * (pow(2, m - 1));

        // if less then N
        if (r < n)
```

```
            ans = r;

        // increment m to get the next number
        m++;
    }

    return ans;
}

// driver code to check the above condition
int main()
{
    long long n = 7;
    cout << answer(n);
    return 0;
}
```

Java

```
 // java program to find largest number
// smaller than equal to n with m set
// bits then m-1 0 bits.
public class GFG {

    // Returns largest number with
    // m set bits then m-1 0 bits.
    static long answer(long n)
    {

        // Start with 2 bits.
        long m = 2;

        // initial answer is 1 which
        // meets the given condition
        long ans = 1;
        long r = 1;

        // check for all numbers
        while (r < n) {

            // compute the number
            r = ((long)Math.pow(2, m) - 1) *
                ((long)Math.pow(2, m - 1));

            // if less then N
            if (r < n)
                ans = r;
```

```
            // increment m to get
            // the next number
            m++;
        }

        return ans;
    }

    // Driver code
    public static void main(String args[]) {

         long n = 7;
         System.out.println(answer(n));
    }
}

// This code is contributed by Sam007
```

Python3

```
 # Python3 program to find
# largest number smaller
# than equal to n with m
# set bits then m-1 0 bits.
import math

# Returns largest number
# with m set bits then
# m-1 0 bits.
def answer(n):

    # Start with 2 bits.
    m = 2;

    # initial answer is
    # 1 which meets the
    # given condition
    ans = 1;
    r = 1;

    # check for all numbers
    while r < n:

        # compute the number
        r = (int)((pow(2, m) - 1) *
                  (pow(2, m - 1)));

        # if less then N
```

```
        if r < n:
            ans = r;

        # increment m to get
        # the next number
        m = m + 1;
    return ans;

# Driver Code
print(answer(7));

# This code is contributed by mits.
```

C#

```
 // C# program to find largest number
// smaller than equal to n with m set
// bits then m-1 0 bits.
using System;

class GFG {

// Returns largest number with
// m set bits then m-1 0 bits.
static long answer(long n)
{

    // Start with 2 bits.
    long m = 2;

    // initial answer is 1 which
    // meets the given condition
    long ans = 1;
    long r = 1;

    // check for all numbers
    while (r < n) {

        // compute the number
        r = ((long)Math.Pow(2, m) - 1) *
            ((long)Math.Pow(2, m - 1));

        // if less then N
        if (r < n)
            ans = r;

        // increment m to get
        // the next number
```

```
        m++;
    }

    return ans;
}

    // Driver Code
    static public void Main ()
    {
        long n = 7;
        Console.WriteLine(answer(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find largest number
// smaller than equal to n with m set
// bits then m-1 0 bits.

// Returns largest number with m set
// bits then m-1 0 bits.
function answer( $n)
{

    // Start with 2 bits.
    $m = 2;

    // initial answer is 1
    // which meets the
    // given condition
    $ans = 1;
    $r = 1;

    // check for all numbers
    while ($r < $n)
    {

        // compute the number
        $r = (pow(2, $m) - 1) *
             (pow(2, $m - 1));

        // if less then N
        if ($r < $n)
            $ans = $r;
```

```
        // increment m to get
        // the next number
        $m++;
    }

    return $ans;
}

    // Driver Code
    $n = 7;
    echo answer($n);

// This code is contributed by Ajit.
?>
```

Output:

```
6
```

Improved By : vt_m, jit_t, Sam007, Mithun Kumar

Source

https://www.geeksforgeeks.org/largest-number-binary-representation-m-1s-m-1-0s/

Chapter 196

Largest set with bitwise OR equal to n

Largest set with bitwise OR equal to n - GeeksforGeeks

Given an integer n, find the largest possible set of non negative integers with bitwise OR equal to n.

Examples:

```
Input : n = 5
Output : arr[] = [0, 1, 4, 5]
The bitwise OR of 0, 1, 4 and 5 equals 5.
It is not possible to obtain a set larger than this.

Input : n = 8
Output : arr[] = [0, 8]
```

Prerequisite: Maximum subset with bitwise OR equal to k

The difference in the above referenced article and this post is the number of elements to be checked. In the above referenced article, we have an array of n numbers and in this post, we have the entire set of non negative numbers.

Traversing an array was simple with the time complexity of O(N), but traversing the boundless set of non negative numbers is not possible. So how do we limit ourselves to a smaller set of numbers?

The answer lies in the concept used. For any number, x greater than n, the bitwise OR of x and n will never be equal to n.
Hence we only need to traverse from 0 to n to obtain our answer.

The second difference is that there will always be an answer for this question. On the other hand, there was no certainty in the existence of an answer in the above referenced article. This is because we can always include n in the resulting set.

Algorithm:
Traverse the numbers from 0 to n, checking its bitwise OR with n. If the bitwise OR equals n, then include that number in the resulting set.

```
 // CPP Program to find the largest set
// with bitwise OR equal to n
#include <bits/stdc++.h>
using namespace std;

// function to find the largest set with
// bitwise OR equal to n
void setBitwiseORk(int n)
{
    vector<int> v;

    for (int i = 0; i <= n; i++) {

        // If the bitwise OR of n and i
        // is equal to n, then include i
        // in the set
        if ((i | n) == n)
            v.push_back(i);
    }

    for (int i = 0; i < v.size(); i++)
        cout << v[i] << ' ';
}

// Driver Code
int main()
{
    int n = 5;

    setBitwiseORk(n);
    return 0;
}
```

Output:

```
0 1 4 5
```

Time complexity : O(N)

Source

https://www.geeksforgeeks.org/largest-set-bitwise-equal-n/

Chapter 197

Left Shift and Right Shift Operators in C/C++

Left Shift and Right Shift Operators in C/C++ - GeeksforGeeks

<< (left shift) Takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.

```
 /* C++ Program to demonstrate use of left shift
   operator */
#include<stdio.h>
int main()
{
    // a = 5(00000101), b = 9(00001001)
    unsigned char a = 5, b = 9;

    // The result is 00001010
    printf("a<<1 = %d\n", a<<1);

    // The result is 00010010
    printf("b<<1 = %d\n", b<<1);
    return 0;
}
```

Output:

```
a<<1 = 10
b<<1 = 18
```

>> (right shift) Takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.

```
 /* C++ Program to demonstrate use of right
   shift operator */
#include<stdio.h>
int main()
{
    // a = 5(00000101), b = 9(00001001)
    unsigned char a = 5, b = 9;

    // The result is 00000010
    printf("a>>1 = %d\n", a>>1);

    // The result is 00000100
    printf("b>>1 = %d\n", b>>1);
    return 0;
}
```

Output:

```
a>>1 = 2
b>>1 = 4
```

Important Points :

- The left shift and right shift operators should not be used for negative numbers. The result of is undefined behaviour if any of the operands is a negative number. For example results of both -1 << 1 and 1 << -1 is undefined.
- If the number is shifted more than the size of integer, the behaviour is undefined. For example, 1 << 33 is undefined if integers are stored using 32 bits. See this for more details.
- The left-shift by 1 and right-shift by 1 are equivalent to multiplication and division by 2 respectively.
  As mentioned in point 1, it works only if numbers are positive.

  ```
   #include<stdio.h>
  int main()
  {
     int x = 19;
     printf ("x << 1 = %d\n", x << 1);
     printf ("x >> 1 = %d\n", x >> 1);
     return 0;
  }
  ```

  Output:

```
x << 1 = 38
x >> 1 = 9
```

- The left-shift of 1 by i is equivalent to 2 raised to power i.
  As mentioned in point 1, it works only if numbers are positive.

```
 #include<stdio.h>
int main()
{
   int i = 3;
   printf("pow(2, %d) = %d\n", i, 1 << i);
   i = 4;
   printf("pow(2, %d) = %d\n", i, 1 << i);
   return 0;
}
```

Output:

```
pow(2, 3) = 8
pow(2, 4) = 16
```

Interesting Facts about Bitwise Operators in C

Improved By : VVNPraveenKumar

Source

https://www.geeksforgeeks.org/left-shift-right-shift-operators-c-cpp/

Chapter 198

Leftover element after performing alternate Bitwise OR and Bitwise XOR operations on adjacent pairs

Leftover element after performing alternate Bitwise OR and Bitwise XOR operations on adjacent pairs - GeeksforGeeks

Given an array of **N(always a power of 2)** elements and **Q** queries.

Every Query consists of two elements **index** and **value.**. We need to write a program that assigns *value* to A_{index} and prints the single element which is left after performing below operations for each query:

- At alternate steps perform **bitwise OR** and **bitwise XOR** operations to the adjacent elements.
- In first iteration select, select n/2 pairs moving from left to right, and do a bitwise OR of all the pair values. In second iteration select (n/2)/2 leftover pairs and do a bitwise XOR on them. In the third iteration select, select ((n/2)/2)/2 leftover pairs moving from left to right, and do a bitwise OR of all the pair values.
- Continue the above steps till we are left with a single element.

Examples:

```
Input : n = 4   m = 2
        arr = [1, 4, 5, 6]
        Queries-
        1st: index=0 value=2
```

```
        2nd: index=3 value=5
Output : 1
         3

Explanation:

1st query:

Assigning 2 to index 0, the sequence is now
[2, 4, 5, 6].
1st iteration: There are 4/2=2 pairs (2, 4) and (5, 6)
2 OR 4 gives 6, and 5 OR 6 gives us 7. So the
sequence is now [6, 7].

2nd iteration: There is 1 pair left now (6, 7)
6^7=1.

Hence the last element left is 1 which is the
answer to our first query.

2nd Query:

Assigning 5 to index 3, the sequence is now
[2, 4, 5, 5].
1st iteration: There are 4/2=2 pairs (2, 4) and (5, 5)
2 OR 4 gives 6, and 5 OR 5 gives us 5. So the
sequence is now [6, 5].

2nd iteration: There is 1 pair left now (6, 5)
6^5=3.

Hence the last element left is 3 which is the
answer to our second query.
```

Naive Approach: The naive approach is to perform every step till we are leftover with one element. Using 2-D vector we will store the resultant elements left after every step. V[steps-1][0..size] gives the number of elements at previous step. If the step number is odd, we perform a bitwise OR operation, else a bitwise XOR operation is done. Repeat the steps till we are left over with one element. The last element left will be our answer.

Below is the implementation of the naive approach:

```
 // CPP program to print the Leftover element after
// performing alternate Bitwise OR and Bitwise XOR
// operations to the pairs.
#include <bits/stdc++.h>
using namespace std;
#define N 1000
```

```
int lastElement(int a[],int n)
{
    // count the step number
    int steps = 1;
    vector<int>v[N];

    // if one element is there, it will be the answer
    if (n==1) return a[0];


    // at first step we do a bitwise OR
    for (int i = 0 ; i < n ; i += 2)
        v[steps].push_back(a[i] | a[i+1]);


    // keep on doing bitwise operations till the
    // last element is left
    while (v[steps].size()>1)
    {

        steps += 1;

        // perform operations
        for (int i = 0 ; i < v[steps-1].size(); i+=2)
        {
            // if step is the odd step
            if (steps&1)
                v[steps].push_back(v[steps-1][i] | v[steps-1][i+1]);
            else  // even step
                v[steps].push_back(v[steps-1][i] ^ v[steps-1][i+1]);
        }
    }

    // answer when one element is left
    return v[steps][0];
}

// Driver Code
int main()
{
    int a[] = {1, 4, 5, 6};
    int n = sizeof(a)/sizeof(a[0]);

    // 1st query
    int index = 0;
    int value = 2;
    a[0] = 2;
```

```
    cout << lastElement(a,n) << endl;

    // 2nd query
    index = 3;
    value = 5;
    a[index] = value;
    cout << lastElement(a,n)  << endl;

    return 0;
}
```

Output:

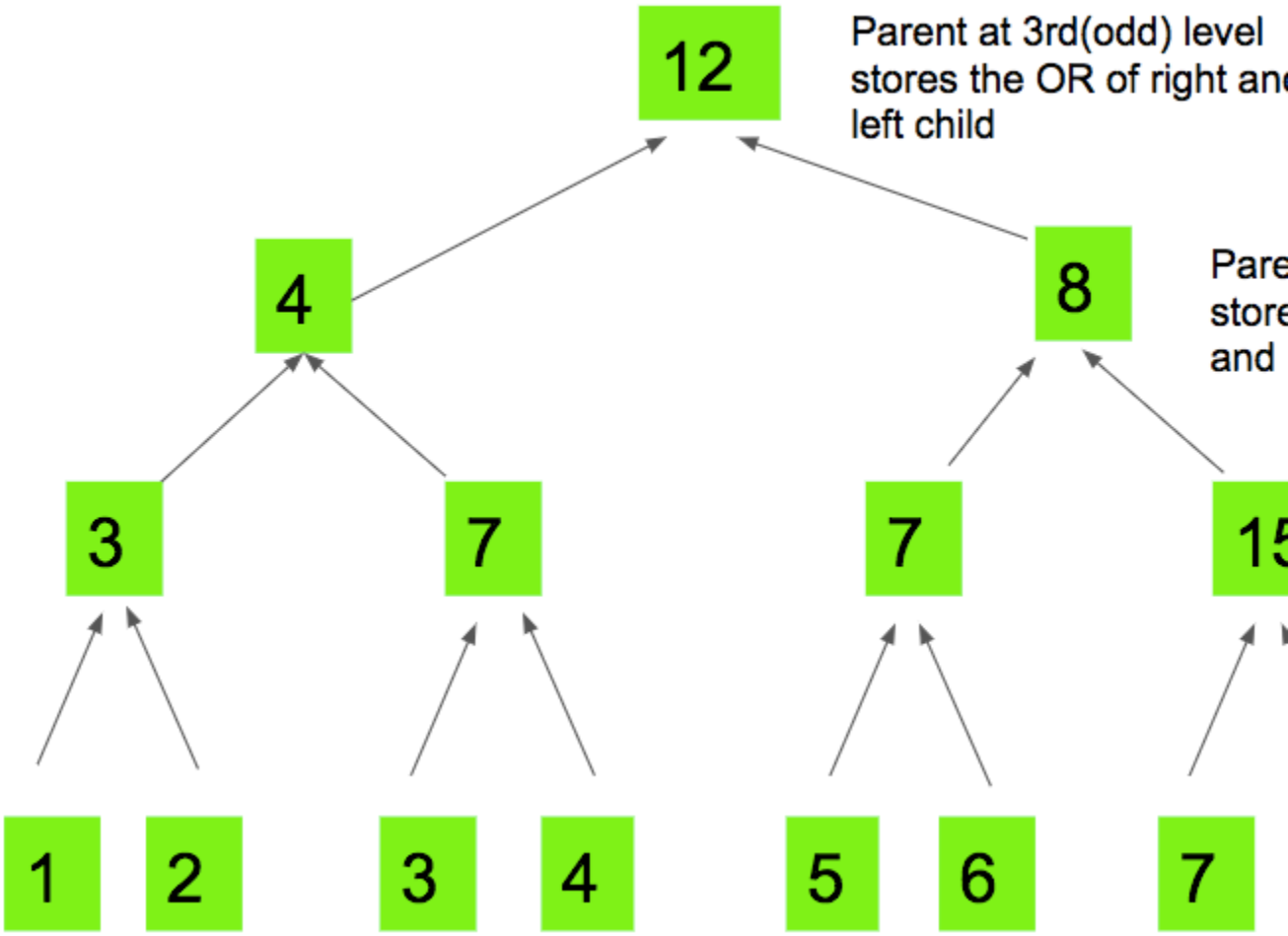
```
1
3
```

Time Complexity: O(N * 2^N)

Efficient Approach: The efficient approach is to use Segment tree. Below is the complete segment tree approach used to solve the problem.

Building the tree

The leaves of the segment tree will store the array values and their parent will store the OR of the leaves. Moving up-ward in the tree, with every alternate step, the parent stores either of bitwise XOR or bitwise OR of left and right child. **At every odd-numbered iteration, we perform the bitwise OR of the pairs and similarly we perform bitwise XOR of pairs at every even-numbered operation**. So the odd-numbered parent will store the bitwise OR of the left and right child. Similarly, the even-numbered parent stores the bitwise XOR of the left and right child. level[] is an array that stores levels of every parent starting from 1, to determine if the pair(right child and left child) below it performs an OR operation or an XOR operation. **The root of the tree will be our answer to the given sequence after every update operation.**



.

The image above explains the construction of the tree if the sequence was [1, 2, 3, 4, 5, 6, 7, 8], then after 3 iterations, we will be left over with 12 which is our answer and is stored at the root.

Answering Query

There is no need to rebuild the complete tree to perform an update operation. To do an update, we should find a **path from the root to the corresponding leaf** and recalculate the values only at the parents that are lying on the found path.

Level of parent:
Using DP on trees, we can easily store the level of every parent. Initialize the leaf nodes level to 0, and keep adding as we move up to every parent.

The recurrence relation for calculating the level of parent is:

level[parent] = level[child] + 1
Here, child is 2*pos + 1 or 2*pos + 2

Below is the implementation of the above approach:

C++

```
 // CPP program to print the Leftover element after
// performing alternate Bitwise OR and
// Bitwise XOR operations to the pairs.
#include <bits/stdc++.h>
using namespace std;
#define N 1000

// array to store the tree
int tree[N];

// array to store the level of every parent
int level[N];

// fucntion to construct the tree
void constructTree(int low, int high, int pos, int a[])
{
    if (low == high)
    {
        // level of child is always 0
        level[pos] = 0;
        tree[pos] = a[high];
        return;
    }
    int mid = (low + high) / 2;

    // recursive call
    constructTree(low, mid, 2 * pos + 1, a);
    constructTree(mid + 1, high, 2 * pos + 2, a);

    // increase the level of every parent, which is
    // level of child + 1
    level[pos] = level[2 * pos + 1] + 1;

    // if the parent is at odd level, then do a
    // bitwise OR
    if (level[pos] & 1)
        tree[pos] = tree[2 * pos + 1] | tree[2 * pos + 2];

    // if the parent is at even level, then
    // do a bitwise XOR
    else
        tree[pos] = tree[2 * pos + 1] ^ tree[2 * pos + 2];
}
```

```
// fucntion that updates the tree
void update(int low, int high, int pos, int index, int a[])
{
    // if it is a leaf and the leaf which is
    // to be updated
    if (low == high and low == index)
    {
        tree[pos] = a[low];
        return;
    }

    // out of range
    if (index < low || index > high)
        return;

    // not a leaf then recurse
    if (low != high)
    {
        int mid = (low + high) / 2;

        // recursive call
        update(low, mid, 2 * pos + 1, index, a);
        update(mid + 1, high, 2 * pos + 2, index, a);

        // check if the parent is at odd or even level
        // and perform OR or XOR according to that
        if (level[pos] & 1)
            tree[pos] = tree[2 * pos + 1] | tree[2 * pos + 2];
        else
            tree[pos] = tree[2 * pos + 1] ^ tree[2 * pos + 2];
    }
}

// fucntion that assigns value to a[index]
// and calls update function to update the tree
void updateValue(int index, int value, int a[], int n)
{
    a[index] = value;
    update(0, n - 1, 0, index, a);
}

// Driver Code
int main()
{
    int a[] = { 1, 4, 5, 6 };
    int n = sizeof(a) / sizeof(a[0]);

    // builds the tree
```

```
    constructTree(0, n - 1, 0, a);

    // 1st query
    int index = 0;
    int value = 2;
    updateValue(index, value, a, n);
    cout << tree[0] << endl;

    // 2nd query
    index = 3;
    value = 5;
    updateValue(index, value, a, n);
    cout << tree[0] << endl;

    return 0;
}
```

Java

```
 // java program to print the Leftover
// element after performing alternate
// Bitwise OR and Bitwise XOR operations
// to the pairs.
import java .io.*;

public class GFG {

    static int N = 1000;

    // array to store the tree
    static int []tree = new int[N];

    // array to store the level of
    // every parent
    static int []level = new int[N];

    // fucntion to construct the tree
    static void constructTree(int low, int high,
                              int pos, int []a)
    {
        if (low == high)
        {

            // level of child is
            // always 0
            level[pos] = 0;
            tree[pos] = a[high];
            return;
```

```
    }
    int mid = (low + high) / 2;

    // recursive call
    constructTree(low, mid, 2 * pos + 1, a);

    constructTree(mid + 1, high,
                          2 * pos + 2, a);

    // increase the level of every parent,
    // which is level of child + 1
    level[pos] = level[2 * pos + 1] + 1;

    // if the parent is at odd level, then
    // do a bitwise OR
    if ((level[pos] & 1) > 0)
        tree[pos] = tree[2 * pos + 1] |
                          tree[2 * pos + 2];

    // if the parent is at even level, then
    // do a bitwise XOR
    else
        tree[pos] = tree[2 * pos + 1] ^
                          tree[2 * pos + 2];
}

// fucntion that updates the tree
static void update(int low, int high, int pos,
                           int index, int []a)
{

    // if it is a leaf and the leaf which is
    // to be updated
    if (low == high && low == index)
    {
        tree[pos] = a[low];
        return;
    }

    // out of range
    if (index < low || index > high)
        return;

    // not a leaf then recurse
    if (low != high)
    {
        int mid = (low + high) / 2;
```

```
        // recursive call
        update(low, mid, 2 * pos + 1, index, a);

        update(mid + 1, high, 2 * pos + 2,
                                    index, a);

        // check if the parent is at odd or
        // even level and perform OR or XOR
        // according to that
        if ((level[pos] & 1) > 0)
            tree[pos] = tree[2 * pos + 1] |
                            tree[2 * pos + 2];
        else
            tree[pos] = tree[2 * pos + 1] ^
                            tree[2 * pos + 2];
    }
}

// fucntion that assigns value to a[index]
// and calls update function to update the
// tree
static void updateValue(int index, int value,
                             int []a, int n)
{
    a[index] = value;
    update(0, n - 1, 0, index, a);
}

// Driver Code
static public void main (String[] args)
{
    int []a = { 1, 4, 5, 6 };
    int n = a.length;;

    // builds the tree
    constructTree(0, n - 1, 0, a);

    // 1st query
    int index = 0;
    int value = 2;
    updateValue(index, value, a, n);
    System.out.println(tree[0]);

    // 2nd query
    index = 3;
    value = 5;
    updateValue(index, value, a, n);
    System.out.println(tree[0]);
```

```
    }
}

// This code is contributed by vt_m.
```

C#

```
 // C# program to print the Leftover
// element after performing alternate
// Bitwise OR and Bitwise XOR
// operations to the pairs.
using System;

public class GFG {

    static int N = 1000;

    // array to store the tree
    static int []tree = new int[N];

    // array to store the level of
    // every parent
    static int []level = new int[N];

    // fucntion to construct the
    // tree
    static void constructTree(int low, int high,
                              int pos, int []a)
    {
        if (low == high)
        {

            // level of child is always 0
            level[pos] = 0;
            tree[pos] = a[high];
            return;
        }
        int mid = (low + high) / 2;

        // recursive call
        constructTree(low, mid, 2 * pos + 1, a);

        constructTree(mid + 1, high,
                               2 * pos + 2, a);

        // increase the level of every parent,
        // which is level of child + 1
        level[pos] = level[2 * pos + 1] + 1;
```

```
    // if the parent is at odd level,
    // then do a bitwise OR
    if ((level[pos] & 1) > 0)
        tree[pos] = tree[2 * pos + 1] |
                        tree[2 * pos + 2];

    // if the parent is at even level,
    // then do a bitwise XOR
    else
        tree[pos] = tree[2 * pos + 1] ^
                       tree[2 * pos + 2];
}

// fucntion that updates the tree
static void update(int low, int high,
           int pos, int index, int []a)
{

    // if it is a leaf and the leaf
    // which is to be updated
    if (low == high && low == index)
    {
        tree[pos] = a[low];
        return;
    }

    // out of range
    if (index < low || index > high)
        return;

    // not a leaf then recurse
    if (low != high)
    {
        int mid = (low + high) / 2;

        // recursive call
        update(low, mid, 2 * pos + 1,
                                index, a);

        update(mid + 1, high, 2 * pos + 2,
                                 index, a);

        // check if the parent is at odd
        // or even level and perform OR
        // or XOR according to that
        if ((level[pos] & 1) > 0)
            tree[pos] = tree[2 * pos + 1] |
```

```
                                tree[2 * pos + 2];
            else
                tree[pos] = tree[2 * pos + 1]
                            ^ tree[2 * pos + 2];
        }
    }

    // fucntion that assigns value to a[index]
    // and calls update function to update
    // the tree
    static void updateValue(int index, int value,
                                  int []a, int n)
    {
        a[index] = value;
        update(0, n - 1, 0, index, a);
    }

    // Driver Code
    static public void Main ()
    {
        int []a = { 1, 4, 5, 6 };
        int n = a.Length;;

        // builds the tree
        constructTree(0, n - 1, 0, a);

        // 1st query
        int index = 0;
        int value = 2;
        updateValue(index, value, a, n);
        Console.WriteLine(tree[0]);

        // 2nd query
        index = 3;
        value = 5;
        updateValue(index, value, a, n);
        Console.WriteLine(tree[0]);
    }
}

// This code is contributed by vt_m.
```

Output:

```
1
3
```

Time Complexity:

- Tree construction: O(N)
- Answering Query: O($\log_2$N)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/leftover-element-performing-alternate-bitwise-bitwise-xor-operations-adjacent-pairs

Chapter 199

Length of the Longest Consecutive 1s in Binary Representation

Length of the Longest Consecutive 1s in Binary Representation - GeeksforGeeks

Given a number n, find length of the longest consecutive 1s in its binary representation.

Examples :

```
Input : n = 14
Output : 3
The binary representation of 14 is 1110.

Input : n = 222
Output : 4
The binary representation of 222 is 11011110.
```

Naive Approach: One simple way would be to simply loop over the bits, and keep track of the number of consecutive set bits, and the maximum that this value has reached. In this approach, we need to convert it to binary (base-2) representation and then find and print the result.

Using Bit Magic: The idea is based on the concept that if we AND a bit sequence with a shifted version of itself, we're effectively removing the trailing 1 from every sequence of consecutive 1s.

```
  11101111   (x)
& 11011110   (x << 1)
----------
```

```
      11001110   (x & (x << 1))
        ^    ^
        |    |
   trailing 1 removed
```

So the operation x = (x & (x << 1)) reduces length of every sequence of 1s by one in binary representation of x. If we keep doing this operation in a loop, we end up with x = 0. The number of iterations required to reach 0 is actually length of the longest consecutive sequence of 1s.

C++

```
 // C++ program to find length of the longest
// consecutive 1s in binary reprsentation of
// a number.
#include<bits/stdc++.h>
using namespace std;

int maxConsecutiveOnes(int x)
{
    // Initialize result
    int count = 0;

    // Count the number of iterations to
    // reach x = 0.
    while (x!=0)
    {
        // This operation reduces length
        // of every sequence of 1s by one.
        x = (x & (x << 1));

        count++;
    }

    return count;
}

// Driver code
int main()
{
    cout << maxConsecutiveOnes(14) << endl;
    cout << maxConsecutiveOnes(222) << endl;
    return 0;
}
```

Java

```
 // Java program to find length of the longest
```

```
// consecutive 1s in binary reprsentation of
// a number.
class MaxConsecutiveOnes
{
    private static int maxConsecutiveOnes(int x)
    {
        // Initialize result
        int count = 0;

        // Count the number of iterations to
        // reach x = 0.
        while (x!=0)
        {
            // This operation reduces length
            // of every sequence of 1s by one.
            x = (x & (x << 1));

            count++;
        }

        return count;
    }

    // Driver code
    public static void main(String strings[])
    {
        System.out.println(maxConsecutiveOnes(14));
        System.out.println(maxConsecutiveOnes(222));
    }
}
```

Python3

```
 # Python program to find
# length of the longest
# consecutive 1s in
# binary reprsentation of
# a number.

def maxConsecutiveOnes(x):

    # Initialize result
    count = 0

    # Count the number of iterations to
    # reach x = 0.
    while (x!=0):
```

```
        # This operation reduces length
        # of every sequence of 1s by one.
        x = (x & (x << 1))

        count=count+1

    return count

# Driver code

print(maxConsecutiveOnes(14))
print(maxConsecutiveOnes(222))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to find length of the
// longest consecutive 1s in binary
// reprsentation of a number.
using System;

class GFG {

    // Function to find length of the
    // longest consecutive 1s in binary
    // reprsentation of a number
    private static int maxConsecutiveOnes(int x)
    {

        // Initialize result
        int count = 0;

        // Count the number of iterations
        // to reach x = 0.
        while (x != 0)
        {

            // This operation reduces length
            // of every sequence of 1s by one.
            x = (x & (x << 1));

            count++;
        }

        return count;
    }
```

```
    // Driver code
    public static void Main()
    {
        Console.WriteLine(maxConsecutiveOnes(14));
        Console.Write(maxConsecutiveOnes(222));
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP program to find length
// of the longest consecutive
// 1s in binary reprsentation of
// a number.

function maxConsecutiveOnes($x)
{
    // Initialize result
    $count = 0;

    // Count the number of
    // iterations to reach x = 0.
    while ($x != 0)
    {
        // This operation reduces
        // length of every sequence
        // of 1s by one.
        $x = ($x & ($x << 1));

        $count++;
    }

    return $count;
}

// Driver code
echo maxConsecutiveOnes(14), "\n";
echo maxConsecutiveOnes(222), "\n";

// This code is contributed by Ajit
?>
```

Output :

```
3
4
```

Improved By : nitin mittal, jit_t

Source

https://www.geeksforgeeks.org/length-longest-consecutive-1s-binary-representation/

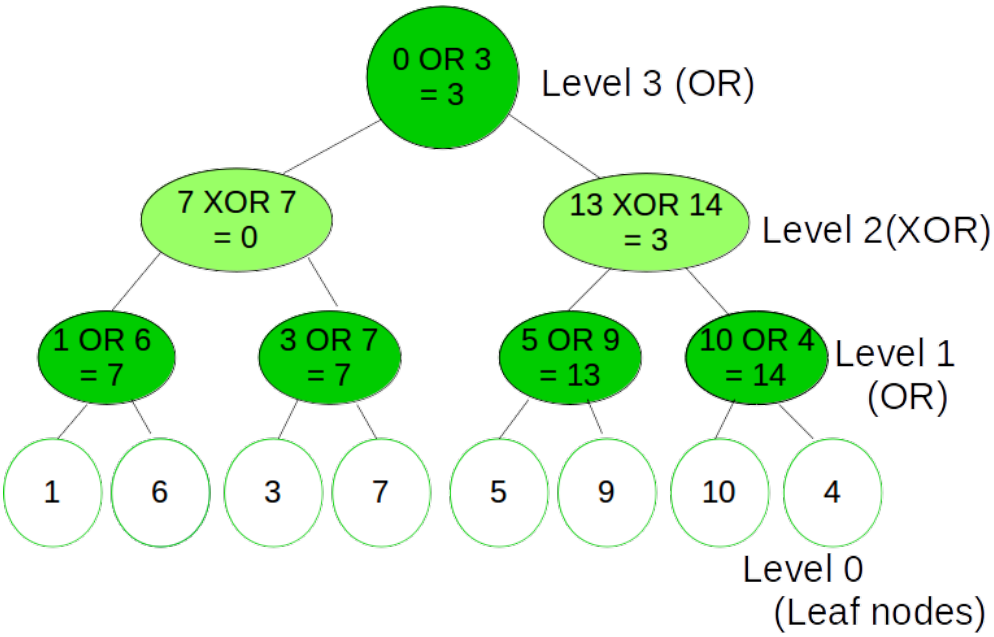
Chapter 200

Levelwise Alternating OR and XOR operations in Segment Tree

Levelwise Alternating OR and XOR operations in Segment Tree - GeeksforGeeks

A Levelwise OR/XOR alternating segment tree is a segment tree, such that at every level the operations OR and XOR alternate. In other words at Level 1 the left and right sub-trees combine together by the OR operation i.e Parent node = Left Child **OR** Right Child and on Level 2 the left
and right sub-trees combine together by the XOR operation i.e Parent node = Left Child **XOR** Right Child

Such a type of Segment tree has the following type of structure:



The operations (OR) and (XOR) indicate which operation was carried out to merge the child node

Given 2^N leaf nodes, the task is to build such a segment tree and print the root node

Examples:

```
Input : Leaves = {1, 6, 3, 7, 5, 9, 10, 4}
Output : Value at Root Node = 3
Explanation : The image given above shows the
segment tree corresponding to the given
set leaf nodes.
```

This is a Extension to the Classical Segment Tree where we represent each node as as integer and the parent node is built by first building the left and the right subtrees and then combining the results of the left and the right children into the parent node. This merging of the left and right children into the parent node is done by a consistent operations For example, MIN()/MAX() in Range Minimum Queries, Sum, XOR, OR, AND, etc. By consistent operations, we mean that this operation is performed to merge any node's left and right child into the parent node by carrying the operation say OP on their results, where OP is a consistent operation.

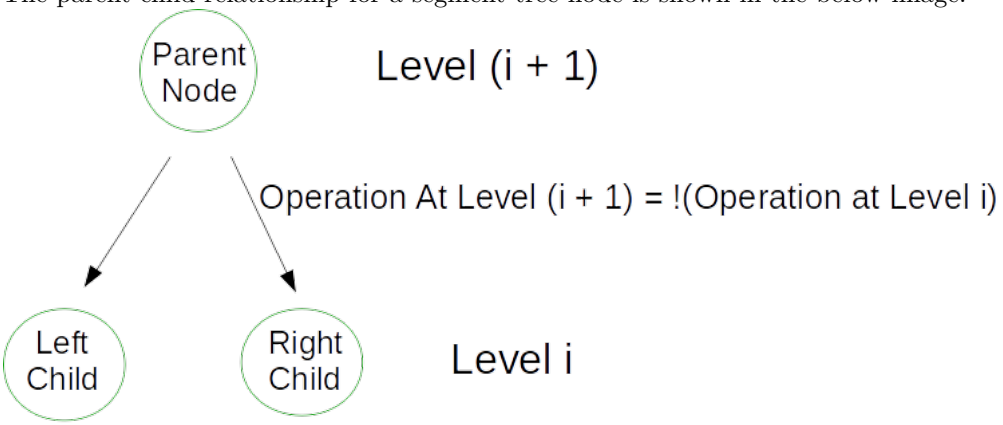
In this Segment tree, we carry two operations that are -: **OR and XOR**.

Now we build the Segment tree in a similar fashion as we do in the classical version, but now when we recursively pass the information for the sub-trees we will also pass information regarding the operation to be carried out at that level since these operations alternate levelwise. Its important to note that a parent node when calls its left and right children the same operation information is passed to both the children as they are on the same level.

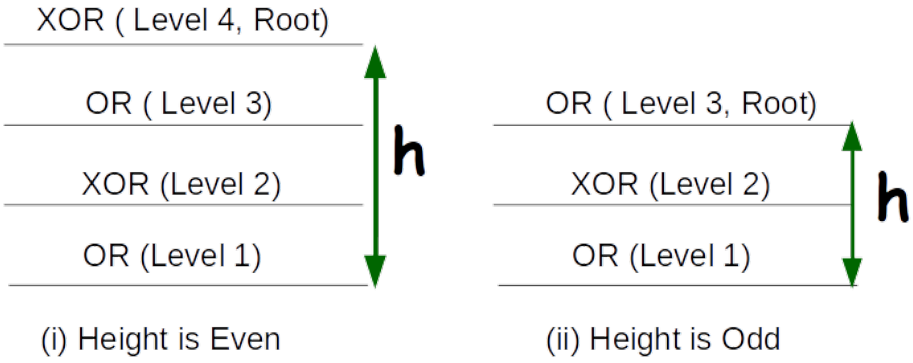
Lets represent the two operations i.e OR and XOR by 0 and 1 respectively. Then if at Level i we have an OR operation the at Level (i + 1) we will have an XOR operation. There if Level i has 0 as operation then level (i + 1) will have 1 as operation

```
Operation at Level (i + 1) = ! (Operation at Level i)
where,
Operation at Level i   {0, 1}
```

The parent child relationship for a segment tree node is shown in the below image:



Now, we need to look at the operation to be carried out at root node.



Note: 'h' denotes height of the segment tree

If we carefully look at the image then it would be easy to figure out that if the height of the tree is even then the root node is a result of XOR operation of its left and right children else a result of OR operation.

```
 // C++ program to build levelwise OR/XOR alternating
// Segment tree
#include <bits/stdc++.h>
```

```
using namespace std;

// A utility function to get the middle index from
// corner indexes.
int getMid(int s, int e) { return s + (e - s) / 2; }

// A recursive function that constructs Segment Tree
// for array[ss..se].
// si is index of current node in segment tree st
// operation denotes which operation is carried out
// at that level to merge the left and right child.
// It's either 0 or 1.
void constructSTUtil(int arr[], int ss, int se, int* st,
                                  int si, int operation)
{
    // If there is one element in array, store it
    // in current node of segment tree and return
    if (ss == se) {
        st[si] = arr[ss];
        return;
    }

    // If there are more than one elements, then
    // recur for left and right subtrees and store
    // the sum of values in this node
    int mid = getMid(ss, se);

    // Build the left and the right subtrees by
    // using the fact that operation at level
    // (i + 1) = ! (operation at level i)
    constructSTUtil(arr, ss, mid, st, si * 2 + 1, !operation);
    constructSTUtil(arr, mid + 1, se, st, si * 2 + 2, !operation);

    // merge the left and right subtrees by checking
    // the operation to  be carried. If operation = 1,
    // then do OR else XOR
    if (operation == 1) {

        // OR operation
        st[si] = (st[2 * si + 1] | st[2 * si + 2]);
    }
    else {

        // XOR operation
        st[si] = (st[2 * si + 1] ^ st[2 * si + 2]);
    }
}
```

```
/* Function to construct segment tree from given array.
   This function allocates memory for segment tree and
   calls constructSTUtil() to fill the allocated memory */
int* constructST(int arr[], int n)
{
    // Allocate memory for segment tree

    // Height of segment tree
    int x = (int)(ceil(log2(n)));

    // Maximum size of segment tree
    int max_size = 2 * (int)pow(2, x) - 1;

    // Allocate memory
    int* st = new int[max_size];

    // operation = 1(XOR) if Height of tree is
    // even else it is 0(OR) for the root node
    int operationAtRoot = (x % 2 == 0 ? 0 : 1);

    // Fill the allocated memory st
    constructSTUtil(arr, 0, n - 1, st, 0, operationAtRoot);

    // Return the constructed segment tree
    return st;
}

// Driver Code
int main()
{

    // leaf nodes
    int leaves[] = { 1, 6, 3, 7, 5, 9, 10, 4 };

    int n = sizeof(leaves) / sizeof(leaves[0]);

    // Build the segment tree
    int* segmentTree = constructST(leaves, n);

    // Root node is at index 0 considering
    // 0-based indexing in segment Tree
    int rootIndex = 0;

    // print value at rootIndex
    cout << "Value at Root Node = " << segmentTree[rootIndex];
}
```

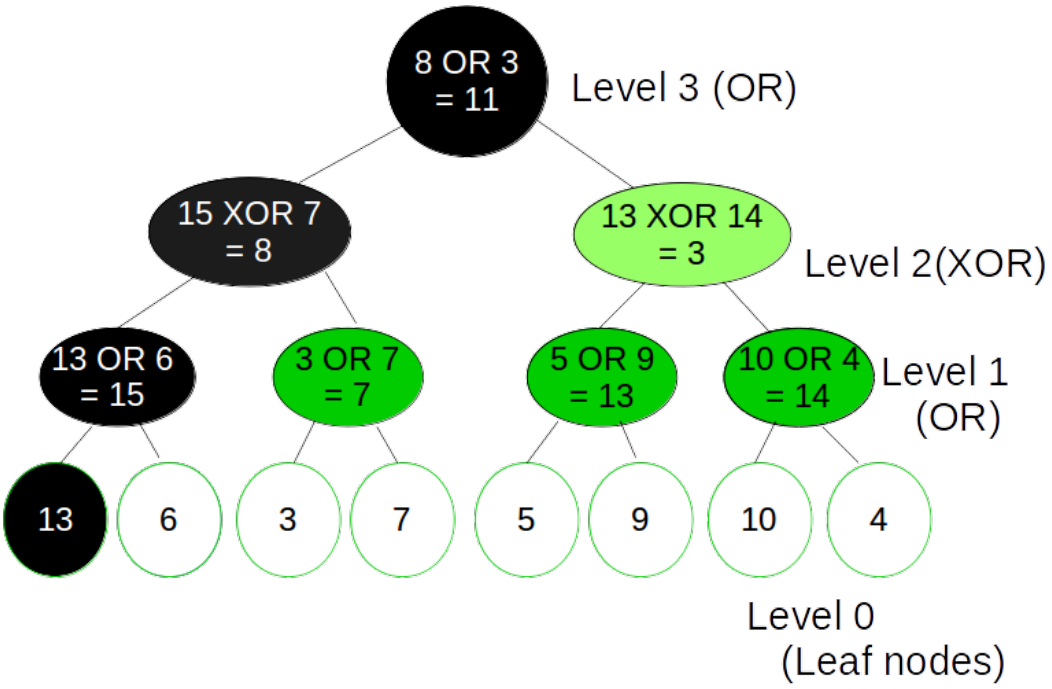
Output:

```
Value at Root Node = 3
```

Time Complexity for tree construction is O(n). There are total 2n-1 nodes, and value of every node is calculated only once in tree construction.

We can also perform Point Updates in a similar manner. If we get an update to update the leaf at index i of the array leaves then we traverse down the tree to the leaf node and perform the update. While coming back to the root node we build the tree again similar to the build() function by passing the operation to be performed at every level and storing the result of applying that operation on values of its left and right children and storing the result into that node.

Consider the following Segment tree after performing the update,
Leaves[0] = 13
Now the updated segment tree looks like this:



Here the nodes in black denote the fact that they were updated

```
 // C++ program to build levelwise OR/XOR alternating
// Segment tree (with updates)
#include <bits/stdc++.h>

using namespace std;

// A utility function to get the middle index from
// corner indexes.
```

```
int getMid(int s, int e) { return s + (e - s) / 2; }

// A recursive function that constructs Segment Tree
// for array[ss..se].
// si is index of current node in segment tree st
// operation denotes which operation is carried out
// at that level to merge the left and right child.
// Its either 0 or 1.
void constructSTUtil(int arr[], int ss, int se, int* st,
                                    int si, int operation)
{
    // If there is one element in array, store it in
    // current node of segment tree and return
    if (ss == se) {
        st[si] = arr[ss];
        return;
    }

    // If there are more than one elements, then recur
    // for left and right subtrees and store the sum of
    // values in this node
    int mid = getMid(ss, se);

    // Build the left and the right subtrees by using
    // the fact that operation at level (i + 1) = !
    // (operation at level i)
    constructSTUtil(arr, ss, mid, st, si * 2 + 1, !operation);
    constructSTUtil(arr, mid + 1, se, st, si * 2 + 2, !operation);

    // merge the left and right subtrees by checking
    // the operation to be carried. If operation = 1,
    // then do OR else XOR
    if (operation == 1) {

        // OR operation
        st[si] = (st[2 * si + 1] | st[2 * si + 2]);
    }
    else {
        // XOR operation
        st[si] = (st[2 * si + 1] ^ st[2 * si + 2]);
    }
}

/* A recursive function to update the nodes which have the given
   index in their range. The following are parameters
    st, si, ss and se are same as getSumUtil()
    i    --> index of the element to be updated. This index is
             in input array.
```

```
   val --> Value to be assigned to arr[i] */
void updateValueUtil(int* st, int ss, int se, int i,
                     int val, int si, int operation)
{
    // Base Case: If the input index lies outside
    //  this segment
    if (i < ss || i > se)
        return;

    // If the input index is in range of this node,
    // then update the value of the node and its
    // children

    // leaf node
    if (ss == se && ss == i) {
        st[si] = val;
        return;
    }

    int mid = getMid(ss, se);

    // Update the left and the right subtrees by
    // using the fact that operation at level
    // (i + 1) = ! (operation at level i)
    updateValueUtil(st, ss, mid, i, val, 2 * si + 1, !operation);
    updateValueUtil(st, mid + 1, se, i, val, 2 * si + 2, !operation);

    // merge the left and right subtrees by checking
    // the operation to to be carried. If operation = 1,
    // then do OR else XOR
    if (operation == 1) {

        // OR operation
        st[si] = (st[2 * si + 1] | st[2 * si + 2]);
    }
    else {

        // XOR operation
        st[si] = (st[2 * si + 1] ^ st[2 * si + 2]);
    }
}

// The function to update a value in input array and
// segment tree. It uses updateValueUtil() to update the
// value in segment tree
void updateValue(int arr[], int* st, int n, int i, int new_val)
{
    // Check for erroneous input index
```

```
    if (i < 0 || i > n - 1) {
        printf("Invalid Input");
        return;
    }

    // Height of segment tree
    int x = (int)(ceil(log2(n)));

    // operation = 1(XOR) if Height of tree is
    // even else it is 0(OR) for the root node
    int operationAtRoot = (x % 2 == 0 ? 0 : 1);

    arr[i] = new_val;

    // Update the values of nodes in segment tree
    updateValueUtil(st, 0, n - 1, i, new_val, 0, operationAtRoot);
}

/* Function to construct segment tree from given array.
   This function allocates memory for segment tree and
   calls constructSTUtil() to fill the allocated memory */
int* constructST(int arr[], int n)
{
    // Allocate memory for segment tree

    // Height of segment tree
    int x = (int)(ceil(log2(n)));

    // Maximum size of segment tree
    int max_size = 2 * (int)pow(2, x) - 1;

    // Allocate memory
    int* st = new int[max_size];

    // operation = 1(XOR) if Height of tree is
    // even else it is 0(OR) for the root node
    int operationAtRoot = (x % 2 == 0 ? 0 : 1);

    // Fill the allocated memory st
    constructSTUtil(arr, 0, n - 1, st, 0, operationAtRoot);

    // Return the constructed segment tree
    return st;
}

// Driver Code
int main()
{
```

```
    int leaves[] = { 1, 6, 3, 7, 5, 9, 10, 4 };
    int n = sizeof(leaves) / sizeof(leaves[0]);

    // Build the segment tree
    int* segmentTree = constructST(leaves, n);

    // Root node is at index 0 considering
    // 0-based indexing in segment Tree
    int rootIndex = 0;

    // print old value at rootIndex
    cout << "Old Value at Root Node = "
         << segmentTree[rootIndex] << endl;

    // perform update leaves[0] = 13
    updateValue(leaves, segmentTree, n, 0, 13);

    cout << "New Value at Root Node = "
         << segmentTree[rootIndex];
    return 0;
}
```

Output:

```
Old Value at Root Node = 3
New Value at Root Node = 11
```

The time complexity of update is also O(Logn). To update a leaf value, we process one node at every level and number of levels is O(Logn).

Improved By : sirjan13

Source

https://www.geeksforgeeks.org/levelwise-alternating-xor-operations-segment-tree/

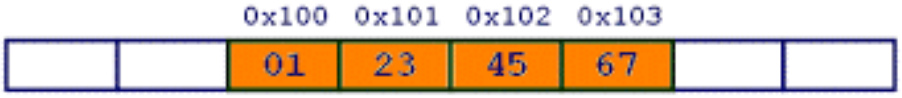
Chapter 201

Little and Big Endian Mystery

Little and Big Endian Mystery - GeeksforGeeks

What are these?
Little and big endian are two ways of storing multibyte data-types (int, float, etc). In little endian machines, last byte of binary representation of the multibyte data-type is stored first. On the other hand, in big endian machines, first byte of binary representation of the multibyte data-type is stored first.

Suppose integer is stored as 4 bytes (For those who are using DOS based compilers such as C++ 3.0 , integer is 2 bytes) then a variable x with value 0x01234567 will be stored as following.



Big Endian

Little Endian

How to see memory representation of multibyte data types on your machine?
Here is a sample C code that shows the byte representation of int, float and pointer.

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

```
/* function to show bytes in memory, from location start to start+n*/
void show_mem_rep(char *start, int n)
{
    int i;
    for (i = 0; i < n; i++)
         printf(" %.2x", start[i]);
    printf("\n");
}

/*Main function to call above function for 0x01234567*/
int main()
{
   int i = 0x01234567;
   show_mem_rep((char *)&i, sizeof(i));
   getchar();
   return 0;
}
```

When above program is run on little endian machine, gives “67 45 23 01” as output , while if it is run on endian machine, gives “01 23 45 67” as output.

Is there a quick way to determine endianness of your machine?
There are n no. of ways for determining endianness of your machine. Here is one quick way of doing the same.

```
 #include <stdio.h>
int main()
{
   unsigned int i = 1;
   char *c = (char*)&i;
   if (*c)
       printf("Little endian");
   else
       printf("Big endian");
   getchar();
   return 0;
}
```

In the above program, a character pointer c is pointing to an integer i. Since size of character is 1 byte when the character pointer is de-referenced it will contain only first byte of integer. If machine is little endian then *c will be 1 (because last byte is stored first) and if machine is big endian then *c will be 0.

Does endianness matter for programmers?
Most of the times compiler takes care of endianness, however, endianness becomes an issue in following cases.

It matters in network programming: Suppose you write integers to file on a little endian machine and you transfer this file to a big endian machine. Unless there is little endian to big endian transformation, big endian machine will read the file in reverse order. You can find such a practical example here.

Standard byte order for networks is big endian, also known as network byte order. Before transferring data on network, data is first converted to network byte order (big endian).

Sometimes it matters when you are using type casting, below program is an example.

```
 #include <stdio.h>
int main()
{
    unsigned char arr[2] = {0x01, 0x00};
    unsigned short int x = *(unsigned short int *) arr;
    printf("%d", x);
    getchar();
    return 0;
}
```

In the above program, a char array is typecasted to an unsigned short integer type. When I run above program on little endian machine, I get 1 as output, while if I run it on a big endian machine I get 256. To make programs endianness independent, above programming style should be avoided.

What are bi-endians?
Bi-endian processors can run in both modes little and big endian.

What are the examples of little, big endian and bi-endian machines ?
Intel based processors are little endians. ARM processors were little endians. Current generation ARM processors are bi-endian.

Motorola 68K processors are big endians. PowerPC (by Motorola) and SPARK (by Sun) processors were big endian. Current version of these processors are bi-endians.

Does endianness affects file formats?
File formats which have 1 byte as a basic unit are independent of endianness e.g., ASCII files . Other file formats use some fixed endianness forrmat e.g, JPEG files are stored in big endian format.

Which one is better — little endian or big endian?
The term little and big endian came from Gulliver's Travels by Jonathan Swift. Two groups could not agree by which end an egg should be opened -a-the little or the big. Just like the egg issue, there is no technological reason to choose one byte ordering convention over the other, hence the arguments degenerate into bickering about sociopolitical issues. As long as one of the conventions is selected and adhered to consistently, the choice is arbitrary.

Improved By : skbarnwal

Source

https://www.geeksforgeeks.org/little-and-big-endian-mystery/

Chapter 202

Lucky alive person in a circle | Code Solution to sword puzzle

Lucky alive person in a circle | Code Solution to sword puzzle - GeeksforGeeks

Given n people standing in a circle where 1st is having sword, find the luckiest person in the circle, if from 1st soldier who is having a sword each have to kill the next soldier and handover the sword to next soldier, in turn the soldier will kill the adjacent soldier and handover the sword to next soldier such that one soldier remain in this war who is not killed by anyone.

Prerequisite : Puzzle 81 | 100 people in a circle with gun puzzle

Examples :

Input : 5
Output : 3
Explanation :
N = 5
Soldier 1 2 3 4 5 (5 soldiers)
In first go 1 3 5 (remains) as 2 and 4 killed by 1 and 3.
In second go 3 as 5 killed 1 and 3rd kill 5 soldier 3 remains alive.

Input : 100
Output : 73
Explanation :
N = 10
Soldiers 1 2 3 4 5 6 7 8 9 10 (10 soldiers)
In first 1 3 5 7 9 as 2 4 6 8 10 were killed by 1 3 5 7 and 9.
In second 1 5 9 as 9 kill 1 and in turn 5 kill 9th soldier.
In third 5 5th soldiers remain alive

Approach : The idea is to use circular linked list. A circular linked list is made based on number of soldier N. As rule state you have to kill your adjacent soldier and handover

the sword to the next soldier who in turn kill his adjacent soldier and handover sword to the next soldier. So in circular linked list the adjacent soldier are killed and the remaining soldier fights against each other in a circular way and a single soldier survive who is not killed by anyone.

```
 // CPP code to find the luckiest person
#include <bits/stdc++.h>
using namespace std;

// Node structure
struct Node {
    int data;
    struct Node* next;
};

Node *newNode(int data)
{
   Node *node = new Node;
   node->data = data;
   node->next = NULL;
   return node;
}

// Function to find the luckiest person
int alivesol(int Num)
{
    if (Num == 1)
        return 1;

    // Create a single node circular
    // linked list.
    Node *last = newNode(1);
    last->next = last;

    for (int i = 2; i <= Num; i++) {
        Node *temp = newNode(i);
        temp->next = last->next;
        last->next = temp;
        last = temp;
    }

    // Starting from first soldier.
    Node *curr = last->next;

    // condition for evaluating the existence
    // of single soldier who is not killed.
    Node *temp;
    while (curr->next != curr) {
```

```
        temp = curr;
        curr = curr->next;
        temp->next = curr->next;

        // deleting soldier from the circular
        // list who is killed in the fight.
        delete curr;
        temp = temp->next;
        curr = temp;
    }

    // Returning the Luckiest soldier who
    // remains alive.
    int res = temp->data;
    delete temp;

    return res;
}

// Driver code
int main()
{
    int N = 100;
    cout << alivesol(N) << endl;
    return 0;
}
```

Output:

```
73
```

Source

https://www.geeksforgeeks.org/lucky-alive-person-circle/

Chapter 203

M-th smallest number having k number of set bits.

M-th smallest number having k number of set bits. - GeeksforGeeks

Given two non-negative integers **m** and **k**. The problem is to find the **m-th** smallest number having **k** number of set bits.

Constraints: 1 <= m, k.

Examples:

```
Input : m = 4, k = 2
Output : 9
(9)10 = (1001)2, it is the 4th smallest
number having 2 set bits.
```

```
Input : m = 6, k = 4
Output : 39
```

Approach: Following are the steps:

1. Find the smallest number having **k** number of set bits. Let it be **num**, where **num** = (1 << k) – 1.
2. Loop for **m-1** times and each time replace **num** with the next higher number than 'num' having same number of bits as in 'num'. Refer this post to find the required next higher number.
3. Finally return **num**.

```
 // C++ implementation to find the mth smallest
// number having k number of set bits
```

```
#include <bits/stdc++.h>
using namespace std;

typedef unsigned int uint_t;

// function to find the next higher number
// with same number of set bits as in 'x'
uint_t nxtHighWithNumOfSetBits(uint_t x)
{
    uint_t rightOne;
    uint_t nextHigherOneBit;
    uint_t rightOnesPattern;

    uint_t next = 0;

    /* the approach is same as dicussed in
       http://www.geeksforgeeks.org/next-higher-number-with-same-number-of-set-bits/
    */
    if (x) {
        rightOne = x & -(signed)x;

        nextHigherOneBit = x + rightOne;

        rightOnesPattern = x ^ nextHigherOneBit;

        rightOnesPattern = (rightOnesPattern) / rightOne;

        rightOnesPattern >>= 2;

        next = nextHigherOneBit | rightOnesPattern;
    }

    return next;
}

// function to find the mth smallest number
// having k number of set bits
int mthSmallestWithKSetBits(uint_t m, uint_t k)
{
    // smallest number having 'k'
    // number of set bits
    uint_t num = (1 << k) - 1;

    // finding the mth smallest number
    // having k set bits
    for (int i = 1; i < m; i++)
        num = nxtHighWithNumOfSetBits(num);
```

```
    // required number
    return num;
}

// Driver program to test above
int main()
{
    uint_t m = 6, k = 4;
    cout << mthSmallestWithKSetBits(m, k);
    return 0;
}
```

Output:

```
39
```

Time Complexity: O(m)

Source

https://www.geeksforgeeks.org/m-th-smallest-number-k-number-set-bits/

Chapter 204

Maximize a given unsigned number number by swapping bits at it's extreme positions.

Maximize a given unsigned number number by swapping bits at it's extreme positions. - GeeksforGeeks

Given a number maximize it by swapping bits at it's extreme positions i.e. at first and last position, second and second last position and so on.
Examples:

```
Input : 4 (0000...0100)
Output : 536870912 (0010...0000)
In the above example swapped 3rd and 3rd last
bit to maximize the given unsigned number.

Input : 12 (00000...1100)
Output : 805306368 (0011000...0000)

In the above example 3rd and 3rd last bit and
4th and 4th last bit are swapped to maximize
the given unsigned number.
```

Naive Approach:
1. Convert the number into it's bit representation and store it's bit representation in an array.

2. Traverse the array from both ends, if the less significant bit of the bit representation is greater than the more significant bit i.e. if less significant bit is 1 and more significant bit is 0 then swap them else take no action.

3. Convert the obtained binary representation back to the number.

Efficient Approach:
1. Create a copy of the original number because the original number would be modified, iteratively obtain the bits at the extreme positions.

2. If less significant bit is 1 and more significant bit is 0 then swap the bits in the bit from only, continue the process until less significant bit's position is less than more significant bit's position.

3. Display the maximized number.

```
 // C++ program to find maximum number by
// swapping extreme bits.
#include <bits/stdc++.h>
using namespace std;

#define ull unsigned long long int

ull findMax(ull num)
{
    ull num_copy = num;

    /* Traverse bits from both extremes */
    int j = sizeof(unsigned long long int) * 8 - 1;
    int i = 0;
    while (i < j) {

        // Obtaining i-th and j-th bits
        int m = (num_copy >> i) & 1;
        int n = (num_copy >> j) & 1;

        /* Swapping the bits if lesser significant
           is greater than higher significant
           bit and accordingly modifying the number */
        if (m > n) {
            x = (1 << i | 1 << j);
            num = num ^ x;
        }

        i++;
        j--;
    }
    return num;
}

// Driver code
int main()
{
```

```
    ull num = 4;
    cout << findMax(num);
    return 0;
}
```

Output:

```
536870912
```

Source

https://www.geeksforgeeks.org/maximize-given-unsigned-number-number-swapping-bits-extreme-positions/

Chapter 205

Maximize the bitwise OR of an array

Maximize the bitwise OR of an array - GeeksforGeeks

Given an array of N integers. The bitwise OR of all the elements of the array has to be maximized by performing one task. The task is to multiply any element of the array **at-most** k times with a given integer x.

Examples :

> Input: a = {1, 1, 1}, k = 1, x = 2
> Output: 3
> Explanation: Any possible choice of doing one element
> of the array will result the same three numbers 1, 1, 2.
> So, the result is 1 | 1 | 2 = 3.
>
> Input: a = {1, 2, 4, 8}, k = 2, x = 3
> Output: 79

Approach: Precompute the prefix and suffix OR arrays.
In one iteration, multiply an element with x^k and do Bitwise OR it with prefix OR i.e. Bitwise OR of all previous elements and suffix OR i.e. Bitwise OR of all next elements and return the maximum value after all iterations.

C++

```
 // C++ program to maximize the Bitwise
// OR Sum in given array
#include <bits/stdc++.h>
using namespace std;

// Function to maximize the bitwise
// OR sum
```

```
int maxOR(long long arr[], int n, int k, int x)
{
    long long preSum[n + 1], suffSum[n + 1];
    long long res, pow = 1;

    // Compute x^k
    for (int i = 0; i < k; i++)
        pow *= x;

    // Find prefix bitwise OR
    preSum[0] = 0;
    for (int i = 0; i < n; i++)
        preSum[i + 1] = preSum[i] | arr[i];

    // Find suffix bitwise OR
    suffSum[n] = 0;
    for (int i = n - 1; i >= 0; i--)
        suffSum[i] = suffSum[i + 1] | arr[i];

    // Find maximum OR  value
    res = 0;
    for (int i = 0; i < n; i++)
        res = max(res, preSum[i] | (arr[i] * pow) | suffSum[i + 1]);

    return res;
}

// Drivers code
int main()
{
    long long arr[] = { 1, 2, 4, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2, x = 3;

    cout << maxOR(arr, n, k, x) << "\n";

    return 0;
}
```

Java

```
 // Java program to maximize the Bitwise
// OR Sum in given array
import java.io.*;

class GFG {

    // Function to maximize the bitwise OR sum
```

```
    public static long maxOR(long arr[], int n,
                                 int k, int x)
    {
        long preSum[] = new long[n + 1];
        long suffSum[] = new long[n + 1];
        long res = 0, pow = 1;

        // Compute x^k
        for (int i = 0; i < k; i++)
            pow *= x;

        // Find prefix bitwise OR
        preSum[0] = 0;
        for (int i = 0; i < n; i++)
            preSum[i + 1] = preSum[i] | arr[i];

        // Find suffix bitwise OR
        suffSum[n] = 0;
        for (int i = n - 1; i >= 0; i--)
            suffSum[i] = suffSum[i + 1] | arr[i];

        // Find maximum OR value
        res = 0;
        for (int i = 0; i < n; i++)
            res = Math.max(res, preSum[i] |
                (arr[i] * pow) | suffSum[i + 1]);

        return res;
    }

    // Drivers code
    public static void main(String args[])
    {
        long arr[] = { 1, 2, 4, 8 };
        int n = 4;
        int k = 2, x = 3;

        long ans = maxOR(arr, n, k, x);
        System.out.println(ans);
    }
}

// This code is contributed by Jaideep Pyne
```

Python 3

```
 # Pyhton 3 program to maximize the Bitwise
# OR Sum in given array
```

```
# Function to maximize the bitwise
# OR sum
def maxOR(arr, n, k, x):

    preSum = [0] * (n + 1)
    suffSum = [0] * (n + 1)
    pow = 1

    # Compute x^k
    for i in range(0 ,k):
        pow *= x

    # Find prefix bitwise OR
    preSum[0] = 0
    for i in range(0, n):
        preSum[i + 1] = preSum[i] | arr[i]

    # Find suffix bitwise OR
    suffSum[n] = 0
    for i in range(n-1, -1, -1):
        suffSum[i] = suffSum[i + 1] | arr[i]

    # Find maximum OR value
    res = 0
    for i in range(0 ,n):
        res = max(res, preSum[i] |
           (arr[i] * pow) | suffSum[i + 1])

    return res

# Drivers code
arr = [1, 2, 4, 8 ]
n = len(arr)
k = 2
x = 3
print(maxOR(arr, n, k, x))

# This code is contributed by Smitha
```

C#

```
 // C# program to maximize the Bitwise
// OR Sum in given array
using System;

class GFG {
```

```
    // Function to maximize the bitwise OR sum
    public static long maxOR(long []arr, int n,
                                int k, int x)
    {
        long []preSum = new long[n + 1];
        long []suffSum = new long[n + 1];
        long res = 0, pow = 1;

        // Compute x^k
        for (int i = 0; i < k; i++)
            pow *= x;

        // Find prefix bitwise OR
        preSum[0] = 0;
        for (int i = 0; i < n; i++)
            preSum[i + 1] = preSum[i] | arr[i];

        // Find suffix bitwise OR
        suffSum[n] = 0;
        for (int i = n - 1; i >= 0; i--)
            suffSum[i] = suffSum[i + 1] | arr[i];

        // Find maximum OR value
        res = 0;
        for (int i = 0; i < n; i++)
            res = Math.Max(res, preSum[i] |
                (arr[i] * pow) | suffSum[i + 1]);

        return res;
    }

    // Drivers code
    public static void Main()
    {
        long []arr = { 1, 2, 4, 8 };
        int n = 4;
        int k = 2, x = 3;

        long ans = maxOR(arr, n, k, x);
        Console.Write(ans);
    }
}

// This code is contributed by Smitha
```

PHP

```
 <?php
```

```
// PHP program to maximize the
// Bitwise OR Sum in given array

// Function to maximize
// the bitwise OR sum

function maxOR($arr, $n, $k, $x)
{
    $res; $pow = 1;

    // Compute x^k
    for ($i = 0; $i < $k; $i++)
        $pow *= $x;

    // Find prefix bitwise OR
    $preSum[0] = 0;
    for ($i = 0; $i < $n; $i++)
        $preSum[$i + 1] = $preSum[$i] |
                                $arr[$i];

    // Find suffix bitwise OR
    $suffSum[$n] = 0;
    for ($i = $n - 1; $i >= 0; $i--)
        $suffSum[$i] = $suffSum[$i + 1] |
                                 $arr[$i];

    // Find maximum OR value
    $res = 0;
    for ($i = 0; $i < $n; $i++)
        $res = max($res, $preSum[$i] |
                    ($arr[$i] * $pow) |
                     $suffSum[$i + 1]);

    return $res;
}

// Driver Code
$arr = array(1, 2, 4, 8);
$n = sizeof($arr);
$k = 2; $x = 3;

echo maxOR($arr, $n, $k, $x),"\n";

// This code is contributed by jit_t
?>
```

Output :

```
79
```

Improved By : jaideeppyne1997, jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/maximize-the-bitwise-or-of-an-array/

Chapter 206

Maximize the number by rearranging bits

Maximize the number by rearranging bits - GeeksforGeeks

Given an unsigned number, find the maximum number that could be formed by using the bits of the given unsigned number.

Examples :

```
Input : 1 (0000....0001)
Output : 2147483648 (1000....0000)

Input : 7 (0000....0111)
Output : 3758096384 (0111....0000)
```

Method 1 (Simple)
1. Find binary representation of the number using simple decimal to binary representation technique.
2. Count number of set bits in the binary representation equal to 'n'.
3. Create a binary representation with its 'n' most significant bits set to 1.
4. Convert the binary representation back to the number.

C++

```
 // An simple C++ program to find
// minimum number formed by bits of a
// given number.
#include <bits/stdc++.h>
#define ll unsigned int
using namespace std;
```

```
// Returns maximum number formed by
// bits of a given number.
ll maximize(ll a)
{
    // _popcnt32(a) gives number of 1's
    // present in binary representation of a.
    ll n = _popcnt32(a);

    // Set most significant n bits of res.
    ll res = 0;
    for (int i=1; i<=n; i++)
       res |= (1 << (32 - i));

    return res;
}

// Driver function.
int main()
{
    ll a = 1;
    cout << maximize(a) << endl;
    return 0;
}
```

Java

```
 // An simple Java program to find
// minimum number formed by bits
// of a given number.
import java.io.*;

class GFG
{
    private static int _popcnt32(long number)
    {
        int counter = 0;

        while(number > 0)
        {
            if(number % 2 == 1)
            {
                counter++;
            }

            //or number = number >> 1
            number = number / 2;
        }
```

```
        return counter;
    }

    // Returns maximum number formed
    // by bits of a given number.
    static long maximize(long a)
    {
        // _popcnt32(a) gives number
        // of 1's present in binary
        // representation of a.
        int n = _popcnt32(a);

        // Set most significant
        // n bits of res.
        long res = 0;
        for (int i = 1; i <= n; i++)
        res = (int)res | (1 << (32 - i));

        return Math.abs(res);
    }

    // Driver Code
    public static void main(String args[])
    {
        long a = 1;
        System.out.print(maximize(a));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Python3

```
 # An simple Python program to
# find minimum number formed
# by bits of a given number.
def _popcnt32(number) :
    counter = 0

    while(number > 0) :
        if(number % 2 == 1) :
            counter = counter + 1

        # or number = number >> 1
        number = int(number / 2)

    return counter
```

```
# Returns maximum number formed
# by bits of a given number.
def maximize(a) :

    # _popcnt32(a) gives number
    # of 1's present in binary
    # representation of a.
    n = _popcnt32(a)

    # Set most significant
    # n bits of res.
    res = 0
    for i in range(1, n + 1) :
        res = int(res |
                  (1 << (32 - i)))

    return abs(res)

# Driver Code
a = 1
print (maximize(a))

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // An simple C# program to find
// minimum number formed by bits
// of a given number.
using System;

class GFG
{

    // Returns maximum number formed
    // by bits of a given number.
    static long maximize(long a)
    {
        // _popcnt32(a) gives number
        // of 1's present in binary
        // representation of a.
        string binaryString = Convert.ToString(a, 2);
        int n = binaryString.Split(new [] {'0'},
                StringSplitOptions.RemoveEmptyEntries).Length;

        // Set most significant n bits of res.
```

```
        long res = 0;
        for (int i = 1; i <= n; i++)
        res = (int)res | (1 << (32 - i));

        return Math.Abs(res);
    }

    // Driver Code.
    static void Main()
    {
        long a = 1;
        Console.WriteLine(maximize(a));
    }
}
// This code is contributed by
// Manish Shaw(manishshaw1)
```

PHP

```
 <?php
// An simple PHP program to find
// minimum number formed by bits
// of a given number.
function _popcnt32($number)
{
    $counter = 0;

    while($number > 0)
    {
        if($number % 2 == 1)
        {
            $counter++;
        }

        //or number = number >> 1
        $number = intval($number / 2);
    }
    return $counter;
}

// Returns maximum number formed
// by bits of a given number.
function maximize($a)
{
    // _popcnt32(a) gives number
    // of 1's present in binary
    // representation of a.
    $n = _popcnt32($a);
```

```
    // Set most significant
    // n bits of res.
    $res = 0;
    for ($i = 1; $i <= $n; $i++)
    $res = intval($res |
                (1 << (32 - $i)));

    return abs($res);
}

// Driver Code
$a = 1;
echo (maximize($a));

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output:

```
2147483648
```

Method 2 (Efficient)
The idea is to first find a number with n least significant set bits, then shift the number left by 32-n.

C++

```
 // An efficient C++ program to find
// minimum number formed by bits of a
// given number.
#include <bits/stdc++.h>
#define ll unsigned int
using namespace std;

// Returns maximum number formed by
// bits of a given number.
ll maximize(ll a)
{
    // _popcnt32(a) gives number of 1's
    // present in binary representation of a.
    ll n = _popcnt32(a);

    // If all 32 bits are set.
    if (n == 32)
```

```
      return a;

    // find a number witn n least
    // significant set bits.
    ll res = (1 << n) - 1;

    // Now shift result by 32 - n
    return (res << (32 - n)) ;
}

// Driver function.
int main()
{
    ll a = 3;
    cout << maximize(a) << endl;
    return 0;
}
```

Java

```
 // An efficient Java program to
// find minimum number formed
// by bits of a given number.
import java.io.*;

class GFG
{
    static long _popcnt32(long n)
    {
        long count = 0;
        while (n != 0)
        {
            n = n & (n - 1);
            count++;
        }
        return count;
    }

    // Returns maximum number
    // formed by bits of a
    // given number.
    static long maximize(long a)
    {
        // _popcnt32(a) gives number
        // of 1's present in binary
        // representation of a.
        long n = _popcnt32(a);
```

```
        // If along 32
        // bits are set.
        if (n == 32)
        return a;

        // find a number witn
        // n least significant
        // set bits.
        long res = (1 << n) - 1;

        // Now shift result
        // by 32 - n
        return (res << (32 - n)) ;
    }

    // Driver Code
    public static void main(String args[])
    {
        long a = 3;
        System.out.print(maximize(a));
    }
}

// This code is contributed by
// ManishShaw(manishshaw1)
```

C#

```
 // An efficient C# program to
// find minimum number formed
// by bits of a given number.
using System;

class GFG
{
    static long _popcnt32(long n)
    {
        long count = 0;
        while (n != 0)
        {
            n = n & (n - 1);
            count++;
        }
        return count;
    }

    // Returns maximum number
    // formed by bits of a
```

```
    // given number.
    static long maximize(long a)
    {
        // _popcnt32(a) gives number
        // of 1's present in binary
        // representation of a.
        long n = _popcnt32(a);

        // If along 32
        // bits are set.
        if (n == 32)
        return a;

        // find a number witn n
        // least significant set bits.
        long res = (1 << Convert.ToInt32(n)) - 1;

        // Now shift result
        // by 32 - n
        return (res << (32 - Convert.ToInt32(n))) ;
    }

    // Driver Code
    static void Main()
    {
        long a = 3;
        Console.WriteLine(maximize(a));
    }
}

// This code is contributed by
// ManishShaw(manishshaw1)
```

PHP

```
 <?php
// An efficient Java program to
// find minimum number formed
// by bits of a given number.

function _popcnt32($n)
{
    $count = 0;
    while ($n != 0)
    {
        $n = $n & ($n - 1);
        $count++;
    }
```

```
    return $count;
}

// Returns maximum number
// formed by bits of a
// given number.
function maximize($a)
{
    // _popcnt32(a) gives number
    // of 1's present in binary
    // representation of a.
    $n = _popcnt32($a);

    // If a$32
    // bits are set.
    if ($n == 32)
    return $a;

    // find a number witn
    // n least significant
    // set bits.
    $res = (1 << $n) - 1;

    // Now shift result
    // by 32 - n
    return ($res << (32 - $n)) ;
}

// Driver Code
$a = 3;
echo (maximize($a));

// This code is contributed by
// ManishShaw(manishshaw1)
?>
```

Output:

```
3221225472
```

Note: The above codes use GCC specific functions. If we wish to write code for other compilers, we may use Count set bits in an integer.

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/maximize-number-rearranging-bits/

Chapter 207

Maximum 0's between two immediate 1's in binary representation

Maximum 0's between two immediate 1's in binary representation - GeeksforGeeks

Given a number n, the task is to find the maximum 0's between two immediate 1's in binary representation of given n. Return -1 if binary representation contains less than two 1's.

Examples :

```
Input : n = 47
Output: 1
// binary of n = 47 is 101111

Input : n = 549
Output: 3
// binary of n = 549 is 1000100101

Input : n = 1030
Output: 7
// binary of n = 1030 is 10000000110

Input : n = 8
Output: -1
// There is only one 1 in binary representation
// of 8.
```

The idea to solve this problem is to use **shift operator**. We just need to find the position of two immediate 1's in binary representation of n and maximize the difference of these position.

- Return -1 if number is 0 or is a power of 2. In these cases there are less than two 1's in binary representation.
- Initialize variable **prev** with position of first right most 1, it basically stores the position of previously seen 1.
- Now take another variable **cur** which stores the position of immediate 1 just after **prev**.
- Now take difference of **cur** − **prev** − **1**, it will be the number of 0's between to immediate 1's and compare it with previous max value of 0's and update **prev** i.e; prev=cur for next iteration.
- Use auxiliary variable **setBit**, which scans all bits of n and helps to detect if current bits is 0 or 1.
- Initially check if N is 0 or power of 2.

Below is the implementation of the above idea :

C++

```
 // C++ program to find maximum number of 0's
// in binary representation of a number
#include <bits/stdc++.h>
using namespace std;

// Returns maximum 0's between two immediate
// 1's in binary representation of number
int maxZeros(int n)
{
    // If there are no 1's or there is only
    // 1, then return -1
    if (n == 0 || (n & (n - 1)) == 0)
        return -1;

    // loop to find position of right most 1
    // here sizeof int is 4 that means total 32 bits
    int setBit = 1, prev = 0, i;
    for (i = 1; i <= sizeof(int) * 8; i++) {
        prev++;

        // we have found right most 1
        if ((n & setBit) == setBit) {
            setBit = setBit << 1;
            break;
        }

        // left shift setBit by 1 to check next bit
        setBit = setBit << 1;
    }
```

```
    // now loop through for remaining bits and find
    // position of immediate 1 after prev
    int max0 = INT_MIN, cur = prev;
    for (int j = i + 1; j <= sizeof(int) * 8; j++) {
        cur++;

        // if cuurent bit is set, then compare
        // difference of cur - prev -1 with
        // previous maximum number of zeros
        if ((n & setBit) == setBit) {
            if (max0 < (cur - prev - 1))
                max0 = cur - prev - 1;

            // update prev
            prev = cur;
        }
        setBit = setBit << 1;
    }
    return max0;
}

// Driver program to run the case
int main()
{
    int n = 549;

    // Initially check that number must not
    // be 0 and power of 2
    cout << maxZeros(n);
    return 0;
}
```

C#

```
 // C# program to find maximum number of 0's
// in binary representation of a number
using System;

class GFG {

    // Returns maximum 0's between two immediate
    // 1's in binary representation of number
    static int maxZeros(int n)
    {
        // If there are no 1's or there is only
        // 1, then return -1
        if (n == 0 || (n & (n - 1)) == 0)
            return -1;
```

```
        // loop to find position of right most 1
        // here sizeof int is 4 that means total 32 bits
        int setBit = 1, prev = 0, i;
        for (i = 1; i <= sizeof(int) * 8; i++) {
            prev++;

            // we have found right most 1
            if ((n & setBit) == setBit) {
                setBit = setBit << 1;
                break;
            }

            // left shift setBit by 1 to check next bit
            setBit = setBit << 1;
        }

        // now loop through for remaining bits and find
        // position of immediate 1 after prev
        int max0 = int.MinValue, cur = prev;
        for (int j = i + 1; j <= sizeof(int) * 8; j++) {
            cur++;

            // if cuurent bit is set, then compare
            // difference of cur - prev -1 with
            // previous maximum number of zeros
            if ((n & setBit) == setBit) {
                if (max0 < (cur - prev - 1))
                    max0 = cur - prev - 1;

                // update prev
                prev = cur;
            }
            setBit = setBit << 1;
        }
        return max0;
    }

    // Driver program to run the case
    static public void Main()
    {
        int n = 549;

        // Initially check that number must not
        // be 0 and power of 2
        Console.WriteLine(maxZeros(n));
    }
}
```

```
// This code is contributed by vt_m.
```

Output:

```
3
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/maximum-0s-two-immediate-1s-binary-representation/

Chapter 208

Maximum AND value of a pair in an array

Maximum AND value of a pair in an array - GeeksforGeeks

We are given an array of n positive elements. we need to find the maximum AND value generated by any pair of element from the array. AND is bitwise & operator.

Examples:

```
Input : arr[] = {4, 8, 12, 16}
Output : Maximum AND value = 8

Input : arr[] = {4, 8, 16, 2}
Output : Maximum AND value = 0
```

Naive Approach : Basic Approach is same as maximum xor value. We iterate over all the possible pair and calculate the AND value of those all. Pick the largest value among them. Time complexity of this solution is O(n^2).

C++

```
 // CPP Program to find maximum XOR value of a pair
#include<bits/stdc++.h>
using namespace std;


// Function for finding maximum and value pair
int maxAND(int arr[], int n)
{
    int res = 0;
```

```
    for (int i=0; i<n; i++)
       for (int j=i+1; j<n; j++)
          res = max(res, arr[i] & arr[j]);

    return res;
}

// Driver function
int main()
{
    int arr[] = {4, 8, 6, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Maximum AND Value = " << maxAND(arr,n);
    return 0;
}
```

Java

```
 // Java Program to find maximum
// XOR value of a pair
import java.util.*;
import java.lang.*;

public class GfG{

// Function for finding maximum
// and value pair
static int maxAND(int arr[], int n)
{
    int res = 0;
    for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++)
        res = res > ( arr[i] & arr[j]) ?
              res : ( arr[i] & arr[j]);

    return res;
}

// driver function
public static void main(String argc[])
{
    int arr[] = {4, 8, 6, 2};
    int n = arr.length;
    System.out.println("Maximum AND Value = " +
                       maxAND(arr,n));
}
}
```

```
// This code is contributed by Prerna Saini
```

Python3

```
 # Python3 Program to find maximum XOR
# value of a pair

# Function for finding maximum and value pair
def maxAND(arr, n) :
    res = 0

    for i in range(0, n) :
        for j in range(i + 1, n) :
            res = max(res, arr[i] & arr[j])

    return res

# Driver function
arr = [4, 8, 6, 2]
n = len(arr)
print("Maximum AND Value = ", maxAND(arr,n))

# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# Program to find maximum
// XOR value of a pair
using System;

public class GfG
{

// Function for finding maximum
// and value pair
static int maxAND(int []arr, int n)
{
    int res = 0;
    for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++)
        res = res > ( arr[i] & arr[j]) ?
              res : ( arr[i] & arr[j]);

    return res;
}

// Driver code
```

```
public static void Main()
{
    int []arr = {4, 8, 6, 2};
    int n = arr.Length;
    Console.WriteLine("Maximum AND Value = " +
                      maxAND(arr, n));
}
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find maximum
// XOR value of a pair

// Function for finding
// maximum and value pair
function maxAND($arr, $n)
{

    $res = 0;
    for ($i = 0; $i < $n; $i++)
    for ($j = $i + 1; $j < $n; $j++)
        $res = max($res, $arr[$i] &
                         $arr[$j]);

    return $res;
}

    // Driver Code
    $arr = array(4, 8, 6, 2);
    $n = count($arr);
    echo "Maximum AND Value = " , maxAND($arr, $n);

// This code is contributed by vt_m.
?>
```

Output:

```
Maximum AND Value = 4
```

Better Approach : Idea is based on properties of AND operator. AND operation of any two bits result in 1 iff both bits are 1. We start from the MSB and check whether we have minimum of two elements of array having set value. If yes then that MSB will be part of

our solution and be added to result otherwise we will discard that bit. Similary, iterating from MSB to LSB (32 to 1) for bit position we can easily check which bit will be part of our solution and will keep adding all such bits to our solution.
Explanation : Lets consider 1st example of {4, 8, 12, 16} :

step 1: Write Bit-representation of each element :
4 = 100, 8 = 1000, 12 = 1100, 16 = 10000
step 2: Check for 1st MSB , pattern = 0 + 16 = 16. Now 5th bit in 16 is set but no other element has 5-bit as set bit so this will not add up to our RES, still RES = 0 and pattern = 0
step 3: Check 4th bit, pattern = 0 + 8 = 8. Now 8 and 12 both have set bit on 4th bit position so that will add up in our solution, RES = 8 and pattern = 8
step 4: Check 3rd bit, pattern = 8 + 4 = 12. Now only 12 has both bits set bit (same as pattern) so we will discard 3rd bit, RES = 8 and pattern = 8
step 5: Check 2nd bit, pattern = 8 + 2 = 10. No element has set bit same as pattern so we will discard 2nd bit, RES = 8 and pattern = 8
step 4: Check 1st bit, pattern = 8 + 1 = 9. No element has set bit same as pattern so we will discard 1st bit, RES = 8 and pattern = 8

C++

```
 // CPP Program to find maximum XOR value of a pair
#include<bits/stdc++.h>
using namespace std;

// Utility function to check number of elements
// having set msb as of pattern
int checkBit(int pattern, int arr[], int n)
{
    int count = 0;
    for (int i = 0; i < n; i++)
        if ((pattern & arr[i]) == pattern)
            count++;
    return count;
}

// Function for finding maximum and value pair
int maxAND (int arr[], int n)
{
    int res = 0, count;

    // iterate over total of 30bits from msb to lsb
    for (int bit = 31; bit >= 0; bit--)
    {
        // find the count of element having set  msb
        count = checkBit(res | (1 << bit),arr,n);
```

```
        // if count >= 2 set particular bit in result
        if ( count >= 2 )
            res |= (1 << bit);
    }

    return res;
}

// Driver function
int main()
{
    int arr[] = {4, 8, 6, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Maximum AND Value = " << maxAND(arr,n);
    return 0;
}
```

Java

```
 // Java Program to find maximum
// XOR value of a pair
import java.util.*;
import java.lang.*;

public class GfG{

// Utility function to check number of elements
// having set msb as of pattern
static int checkBit(int pattern, int arr[], int n)
{
    int count = 0;
    for (int i = 0; i < n; i++)
        if ((pattern & arr[i]) == pattern)
            count++;
    return count;
}

// Function for finding maximum and value pair
static int maxAND (int arr[], int n)
{
    int res = 0, count;

    // iterate over total of 30bits
    // from msb to lsb
    for (int bit = 31; bit >= 0; bit--)
    {
        // find the count of element
        // having set msb
```

```
        count = checkBit(res | (1 << bit), arr, n);

        // if count >= 2 set particular
        // bit in result
        if ( count >= 2 )
            res |= (1 << bit);
    }

    return res;
}

// driver function
public static void main(String argc[])
{
    int arr[] = {4, 8, 6, 2};
    int n = arr.length;
    System.out.println("Maximum AND Value = " +
                       maxAND(arr, n));
}
}

// This code is contributed by Prerna Saini
```

Python3

```
 # Python3 Program to find maximum XOR
# value of a pair

# Utility function to check number of
# elements having set msb as of pattern
def checkBit(pattern,arr,  n) :
    count = 0

    for i in range(0, n) :
        if ((pattern & arr[i]) == pattern) :
            count = count + 1
    return count

# Function for finding maximum and
# value pair
def maxAND (arr,  n) :
    res = 0

    # iterate over total of 30bits
    # from msb to lsb
    for bit in range(31,-1,-1) :

        # find the count of element
```

```
        # having set  msb
        count = checkBit(res | (1 << bit), arr, n)

        # if count >= 2 set particular
        # bit in result
        if ( count >= 2 ) :
            res =res | (1 << bit)

    return res


# Driver function
arr = [4, 8, 6, 2]
n = len(arr)
print("Maximum AND Value = ", maxAND(arr, n))

# This code is contributed by Nikita Tiwari
```

C#

```
 // C# Program to find maximum
// XOR value of a pair
using System;

public class GfG
{

// Utility function to check
// number of elements having
// set msb as of pattern
static int checkBit(int pattern,
                    int []arr,
                    int n)
{
    int count = 0;
    for (int i = 0; i < n; i++)
        if ((pattern & arr[i]) == pattern)
            count++;
    return count;
}

// Function for finding maximum
// and value pair
static int maxAND (int []arr, int n)
{
    int res = 0, count;

    // iterate over total of 30bits
```

```
    // from msb to lsb
    for (int bit = 31; bit >= 0; bit--)
    {

        // find the count of element
        // having set msb
        count = checkBit(res | (1 << bit), arr, n);

        // if count >= 2 set particular
        // bit in result
        if (count >= 2)
            res |= (1 << bit);
    }

    return res;
}

// Driver Code
public static void Main()
{
    int []arr = {4, 8, 6, 2};
    int n = arr.Length;
    Console.WriteLine("Maximum AND Value = " +
                      maxAND(arr, n));
}
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find maximum
// XOR value of a pair

// Utility function to check
// number of elements having
// set msb as of pattern
function checkBit($pattern, $arr, $n)
{
    $count = 0;
    for ($i = 0; $i < $n; $i++)
        if (($pattern & $arr[$i]) == $pattern)
            $count++;
    return $count;
}

// Function for finding
```

```
// maximum and value pair
function maxAND ($arr, $n)
{
    $res = 0;$count;

    // iterate over total of
    // 30bits from msb to lsb
    for ($bit = 31; $bit >= 0; $bit--)
    {

        // find the count of element
        // having set msb
        $count = checkBit($res | (1 << $bit),
                                  $arr, $n);

        // if count >= 2 set particular
        // bit in result
        if ( $count >= 2 )
            $res |= (1 << $bit);
    }

    return $res;
}

    // Driver Code
    $arr = array(4, 8, 6, 2);
    $n = count($arr);
    echo "Maximum AND Value = " , maxAND($arr,$n);

// This code is contributed by vt_m.
?>
```

Output:

```
Maximum AND Value = 4
```

Time Complexity : O(n)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/maximum-value-pair-array/

Chapter 209

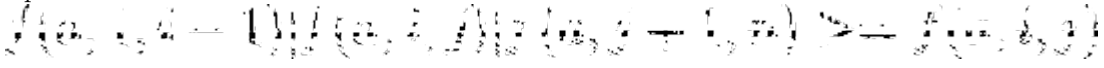
Maximum OR sum of sub-arrays of two different arrays

Maximum OR sum of sub-arrays of two different arrays - GeeksforGeeks

Given two arrays of positive integers. Select two sub-arrays of equal size from each array and calculate maximum possible OR sum of the two sub-arrays.
Note: Let f(x, l, r) is the OR sum of all the elements in the range [l, r] in array x.
Examples :

```
Input : A[] = {1, 2, 4, 3, 2}
        B[] = {2, 3, 3, 12, 1}
Output : 22
Explanation: Here, one way to get maximum
sum is to select sub-array [l = 2, r = 4]
f(A, 2, 4) = 2|4|3 = 7
f(B, 2, 4) = 3|3|12 = 15
So, f(A, 2, 4) + f(B, 2, 4) = 7 + 15 = 22.
This sum can be achieved in many other ways.

Input : A[] = {1, 2, 2}
        B[] = {2, 1, 3}
Output : 6
```

Observe the operation of Bitwise OR operator. If we take two integers X and Y, then (X|Y >= X). It can be proved by taking some examples. Lets derive a formula using the above equation.



and also -

from the above two equations,

So, we get maximum sum when we take the OR of the whole array ->

Below is the implementation of above approach:

C++

```
 // CPP program to find maximum OR sum
#include <bits/stdc++.h>
using namespace std;

// function to find maximum OR sum
void MaximumSum(int a[], int b[], int n)
{
    int sum1 = 0, sum2 = 0;

    // OR sum of all the elements
    // in both arrays
    for (int i = 0; i < n; i++) {
        sum1 |= a[i];
        sum2 |= b[i];
    }
    cout << sum1 + sum2 << endl;
}

// Driver Code
int main()
{
    int A[] = { 1, 2, 4, 3, 2 };
    int B[] = { 2, 3, 3, 12, 1 };
    int n = sizeof(A) / sizeof(A[0]);
    MaximumSum(A, B, n);
    return 0;
}
```

Java

```
 // Java program to find maximum OR sum

class GFG {

// function to find maximum OR sum
static void MaximumSum(int a[], int b[], int n)
```

```
{
    int sum1 = 0, sum2 = 0;

    // OR sum of all the elements
    // in both arrays
    for (int i = 0; i < n; i++) {
    sum1 |= a[i];
    sum2 |= b[i];
    }
    System.out.println(sum1 + sum2);
}

// Driver code
public static void main(String arg[])
{
    int A[] = {1, 2, 4, 3, 2};
    int B[] = {2, 3, 3, 12, 1};
    int n = A.length;
    MaximumSum(A, B, n);
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python 3 program to
# find maximum OR sum

# function to find
# maximum OR sum
def MaximumSum(a, b, n):

    sum1 = 0
    sum2 = 0

    # OR sum of all the
    # elements in both arrays
    for i in range(0, n):
        sum1 |= a[i]
        sum2 |= b[i]

    print(sum1 + sum2)

# Driver Code
A = [ 1, 2, 4, 3, 2 ]
B = [ 2, 3, 3, 12, 1 ]
n = len(A)
```

```
MaximumSum(A, B, n)

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to find maximum OR sum
using System;

class GFG {

    // function to find maximum OR sum
    static void MaximumSum(int []a, int []b, int n)
    {
        int sum1 = 0, sum2 = 0;

        // OR sum of all the elements
        // in both arrays
        for (int i = 0; i < n; i++)
        {
            sum1 |= a[i];
            sum2 |= b[i];
        }
        Console.WriteLine(sum1 + sum2);
    }

    // Driver code
    public static void Main()
    {
        int []A = {1, 2, 4, 3, 2};
        int []B = {2, 3, 3, 12, 1};
        int n = A.Length;
        MaximumSum(A, B, n);
    }
}

// This code is contributed by Vt_m.
```

PHP

```
 <?php
// PHP program to find maximum OR sum

// function to find maximum OR sum
function MaximumSum($a, $b, $n)
{
```

```
    $sum1 = 0;
    $sum2 = 0;

    // OR sum of all the elements
    // in both arrays
    for ($i = 0; $i < $n; $i++)
    {
        $sum1 |= $a[$i];
        $sum2 |= $b[$i];
    }
    echo ($sum1 + $sum2)."\n";
}

// Driver Code
$A = array(1, 2, 4, 3, 2 );
$B = array(2, 3, 3, 12, 1 );
$n = sizeof($A) / sizeof($A[0]);
MaximumSum($A, $B, $n);

// This code is contributed by mits

?>
```

Output :

```
22
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/maximum-sum-sub-arrays-two-different-arrays/

Chapter 210

Maximum XOR using K numbers from 1 to n

Maximum XOR using K numbers from 1 to n - GeeksforGeeks

Given an positive integer n and k. Find maximum xor of 1 to n using at most k numbers. Xor sum of 1 to n is defined as 1 ^ 2 ^ 3 ^ … ^ n.

Examples :

```
Input :  n = 4, k = 3
Output : 7
Explanation
Maximum possible xor sum is 1 ^ 2 ^ 4 = 7.

Input : n = 11, k = 1
Output : 11
Explanation
Maximum Possible xor sum is 11.
```

If we have k = 1 then the maximum possible xor sum is 1. Now for k > 1 we can always have an number with its all bits set to 1. So result will be maximum number greater than n with its all bits set to 1.

C++

```
 // CPP program to find max xor sum
// of 1 to n using atmost k numbers
#include <bits/stdc++.h>
using namespace std;
```

```
// To return max xor sum of 1 to n
// using at most k numbers
int maxXorSum(int n, int k)
{
    // If k is 1 then maximum
    // possible sum is n
    if (k == 1)
        return n;

    // Finding number greater than
    // or equal to n with most significant
    // bit same as n. For example, if n is
    // 4, result is 7. If n is 5 or 6, result
    // is 7
    int res = 1;
    while (res <= n)
        res <<= 1;

    // Return res - 1 which denotes
    // a number with all bits set to 1
    return res - 1;
}

// Driver program
int main()
{
    int n = 4, k = 3;
    cout << maxXorSum(n, k);
    return 0;
}
```

Java

```
 // Java program to find max xor sum
// of 1 to n using atmost k numbers
public class Main {

    // To return max xor sum of 1 to n
    // using at most k numbers
    static int maxXorSum(int n, int k)
    {
        // If k is 1 then maximum
        // possible sum is n
        if (k == 1)
            return n;

        // Finding number greater than
        // or equal to n with most significant
```

```
        // bit same as n. For example, if n is
        // 4, result is 7. If n is 5 or 6, result
        // is 7
        int res = 1;
        while (res <= n)
            res <<= 1;

        // Return res - 1 which denotes
        // a number with all bits set to 1
        return res - 1;
    }

    // Driver program to test maxXorSum()
    public static void main(String[] args)
    {
        int n = 4, k = 3;
        System.out.print(maxXorSum(n, k));
    }
}
```

Python

```
 # Python3 code to find max xor sum
# of 1 to n using atmost k numbers

# To return max xor sum of 1 to n
# using at most k numbers
def maxXorSum( n , k ):
    # If k is 1 then maximum
    # possible sum is n
    if k == 1:
        return n

    # Finding number greater than
    # or equal to n with most significant
    # bit same as n. For example, if n is
    # 4, result is 7. If n is 5 or 6, result
    # is 7
    res = 1
    while res <= n:
        res <<= 1

    # Return res - 1 which denotes
    # a number with all bits set to 1
    return res - 1

# Driver code
n = 4
```

```
k = 3
print( maxXorSum(n, k) )

# This code is contributed by Abhishek Sharma44.
```

C#

```
 // C# program to find max xor sum
// of 1 to n using atmost k numbers
using System;

public class main {

    // To return max xor sum of 1 to n
    // using at most k numbers
    static int maxXorSum(int n, int k)
    {
        // If k is 1 then maximum
        // possible sum is n
        if (k == 1)
            return n;

        // Finding number greater than
        // or equal to n with most significant
        // bit same as n. For example, if n is
        // 4, result is 7. If n is 5 or 6, result
        // is 7
        int res = 1;
        while (res <= n)
            res <<= 1;

        // Return res - 1 which denotes
        // a number with all bits set to 1
        return res - 1;
    }

    // Driver program
    public static void Main()
    {
        int n = 4, k = 3;
        Console.WriteLine(maxXorSum(n, k));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find max xor sum
// of 1 to n using atmost k numbers

// To return max xor sum of 1 to n
// using at most k numbers
function maxXorSum($n, $k)
{
    // If k is 1 then maximum
    // possible sum is n
    if ($k == 1)
        return $n;

    // Finding number greater than
    // or equal to n with most
    // significant bit same as n.
    // For example, if n is 4, result
    // is 7. If n is 5 or 6, result is 7
    $res = 1;
    while ($res <= $n)
        $res <<= 1;

    // Return res - 1 which denotes
    // a number with all bits set to 1
    return $res - 1;
}

// Driver code
$n = 4;
$k = 3;
echo maxXorSum($n, $k);

// This code is contributed by Mithun Kumar
?>
```

Output :

```
7
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/maximum-xor-using-k-numbers-1-n/

Chapter 211

Maximum XOR value of a pair from a range

Maximum XOR value of a pair from a range - GeeksforGeeks

Given a range [L, R], we need to find two integers in this range such that their XOR is maximum among all possible choices of two integers. More Formally,
given [L, R], find max (A ^ B) where L <= A, B
Examples :

```
Input  : L = 8
         R = 20
Output : 31
31 is XOR of 15 and 16.

Input  : L = 1
         R = 3
Output : 3
```

A **simple solution** is to generate all pairs, find their XOR values and finally return the maximum XOR value.

An **efficient solution** is to consider pattern of binary values from L to R. We can see that first bit from L to R either changes from 0 to 1 or it stays 1 i.e. if we take the XOR of any two numbers for maximum value their first bit will be fixed which will be same as first bit of XOR of L and R itself.
After observing the technique to get first bit, we can see that if we XOR L and R, the most significant bit of this XOR will tell us the maximum value we can achieve i.e. let XOR of L and R is 1xxx where x can be 0 or 1 then maximum XOR value we can get is 1111 because from L to R we have all possible combination of xxx and it is always possible to choose these bits in such a way from two numbers such that their XOR becomes all 1. It is explained below with some examples,

```
Examples 1:
L = 8    R = 20
L ^ R = (01000) ^ (10100) = (11100)
Now as L ^ R is of form (1xxxx) we
can get maximum XOR as (11111) by
choosing A and B as 15 and 16 (01111
and 10000)

Examples 2:
L = 16     R = 20
L ^ R = (10000) ^ (10100) = (00100)
Now as L ^ R is of form (1xx) we can
get maximum xor as (111) by choosing
A and B as 19 and 20 (10011 and 10100)
```

So the solution of this problem depends on the value of (L ^ R) only. We will calculate the L^R value first and then from most significant bit of this value, we will add all 1s to get the final result.

C++

```
 // C/C++ program to get maximum xor value
// of two numbers in a range
#include <bits/stdc++.h>
using namespace std;

// method to get maximum xor value in range [L, R]
int maxXORInRange(int L, int R)
{
    // get xor of limits
    int LXR = L ^ R;

    //  loop to get msb position of L^R
    int msbPos = 0;
    while (LXR)
    {
        msbPos++;
        LXR >>= 1;
    }

    // construct result by adding 1,
    // msbPos times
    int maxXOR = 0;
    int two = 1;
    while (msbPos--)
    {
```

```
        maxXOR += two;
        two <<= 1;
    }

    return maxXOR;
}

//  Driver code to test above methods
int main()
{
    int L = 8;
    int R = 20;
    cout << maxXORInRange(L, R) << endl;
    return 0;
}
```

Java

```
 // Java program to get maximum xor value
// of two numbers in a range

class Xor
{
    // method to get maximum xor value in range [L, R]
    static int maxXORInRange(int L, int R)
    {
        // get xor of limits
        int LXR = L ^ R;

        //  loop to get msb position of L^R
        int msbPos = 0;
        while (LXR > 0)
        {
            msbPos++;
            LXR >>= 1;
        }

        // construct result by adding 1,
        // msbPos times
        int maxXOR = 0;
        int two = 1;
        while (msbPos-- >0)
        {
            maxXOR += two;
            two <<= 1;
        }

        return maxXOR;
```

```
    }

    // main function
    public static void main (String[] args)
    {
        int L = 8;
        int R = 20;
        System.out.println(maxXORInRange(L, R));
    }
}
```

Python3

```
 # Python3 program to get maximum xor
# value of two numbers in a range

# Method to get maximum xor
# value in range [L, R]
def maxXORInRange(L, R):

    # get xor of limits
    LXR = L ^ R

    # loop to get msb position of L^R
    msbPos = 0
    while(LXR):

        msbPos += 1
        LXR >>= 1


    # construct result by adding 1,
    # msbPos times
    maxXOR, two = 0, 1

    while (msbPos):

        maxXOR += two
        two <<= 1
        msbPos -= 1

    return maxXOR

# Driver code
L, R = 8, 20
print(maxXORInRange(L, R))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to get maximum xor
// value of two numbers in a range
using System;

class Xor
{

    // method to get maximum xor
    // value in range [L, R]
    static int maxXORInRange(int L, int R)
    {

        // get xor of limits
        int LXR = L ^ R;

        // loop to get msb position of L^R
        int msbPos = 0;
        while (LXR > 0)
        {
            msbPos++;
            LXR >>= 1;
        }

        // construct result by
        // adding 1, msbPos times
        int maxXOR = 0;
        int two = 1;
        while (msbPos-- >0)
        {
            maxXOR += two;
            two <<= 1;
        }

        return maxXOR;
    }

    // Driver code
    public static void Main()
    {
        int L = 8;
        int R = 20;
        Console.WriteLine(maxXORInRange(L, R));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program to get maximum
// xor value of two numbers
// in a range

// method to get maximum xor
// value in range [L, R]
function maxXORInRange($L, $R)
{
    // get xor of limits
    $LXR = $L ^ $R;

    // loop to get msb
    // position of L^R
    $msbPos = 0;
    while ($LXR)
    {
        $msbPos++;
        $LXR >>= 1;
    }

    // construct result by
    // adding 1, msbPos times
    $maxXOR = 0;
    $two = 1;
    while ($msbPos--)
    {
        $maxXOR += $two;
        $two <<= 1;
    }

    return $maxXOR;
}

// Driver Code
$L = 8;
$R = 20;
echo maxXORInRange($L, $R), "\n";

// This code is contributed by aj_36
?>
```

Output :

```
31
```

Improved By : jit_t, shaurya uppal

Source

https://www.geeksforgeeks.org/maximum-xor-value-of-a-pair-from-a-range/

Chapter 212

Maximum XOR-value of at-most k-elements from 1 to n

Maximum XOR-value of at-most k-elements from 1 to n - GeeksforGeeks

You are given two positive integer n and k. You have to calculate the maximum possible XOR value of at most k-elements from 1 to n.
Note:k > 1

Examples :

```
Input : n = 7, k = 3
Output : 7
Explanation : You can select 1, 2, 4 for maximum XOR-value

Input : n = 7, k = 2
Output : 7
Explanation : You can select 3 and 4 for maximum value.
```

For any value of k we can select atleast two numbers from 1 to n and for the required result we have to take a closer look on the bit-representation of n. So lets understand it through an example. Suppose n = 6 and k = 2:
Bit representation of 6 = 110
Bit representation of 5 = 101
Bit representation of 4 = 100
Bit representation of 3 = 011
Bit representation of 2 = 010
Bit representation of 1 = 001

Now, you can see that after selecting as much numbers you want and selecting any of them you can not obtain XOR value greater than 111 i.e 7. So, for a given n and k >1 the maximum possible XOR value is $\mathbf{2^{log2(n)+1}-1}$ (that is the value when all bits of n are turned to 1).

C++

```
 // Program to obtain maximum XOR value sub-array
#include <bits/stdc++.h>
using namespace std;

// function to calculate maximum XOR value
int maxXOR(int n, int k) {
  int c = log2(n) + 1;

  // Return (2^c - 1)
  return ((1 << c) - 1);
}

// driver program
int main() {
  int n = 12;
  int k = 3;
  cout << maxXOR(n, k);
  return 0;
}
```

Java

```
 // Program to obtain maximum
// XOR value sub-array
import java.lang.*;

class GFG
{
// function to calculate
// maximum XOR value
static int maxXOR(int n, int k)
{
int c = (int) (Math.log(n) /
               Math.log(2)) + 1;

// Return (2^c - 1)
return ((1 << c) - 1);
}

// Driver Code
public static void main(String[] args)
{
int n = 12;
int k = 3;
System.out.println(maxXOR(n, k));
```

```
}
}

// This code is contributed by Smitha
```

Python3

```
 # Python3 program to obtain maximum
# XOR value sub-array
import math

# Function to calculate maximum XOR value
def maxXOR(n, k):
    c = int(math.log(n, 2)) + 1

    # Return (2^c - 1)
    return ((1 << c) - 1)

# Driver Code
n = 12; k = 3
print (maxXOR(n, k))

# This code is contributed by shreyanshi_arun.
```

C#

```
 // Program to obtain maximum
// XOR value sub-array
using System;

class GFG
{
// function to calculate
// maximum XOR value
static int maxXOR(int n, int k)
{
int c = (int) (Math.Log(n) /
               Math.Log(2)) + 1;

// Return (2^c - 1)
return ((1 << c) - 1);
}

// Driver Code
public static void Main(String[] args)
{
int n = 12;
```

```
int k = 3;
Console.Write(maxXOR(n, k)) ;
}
}

// This code is contributed by Smitha
```

PHP

```
 <?php
// Program to obtain maximum
// XOR value sub-array

// function to calculate
// maximum XOR value
function maxXOR($n, $k)
{
    $c = log($n, 2) + 1;

    // Return (2^c - 1)
    return ((1 << $c) - 1);
}

// Driver Code
$n = 12;
$k = 3;
echo maxXOR($n, $k);

// This code is contributed by aj_36
?>
```

Output:

```
15
```

Improved By : jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/maximum-xor-value-k-elements-1-n/

Chapter 213

Maximum set bit sum in array without considering adjacent elements

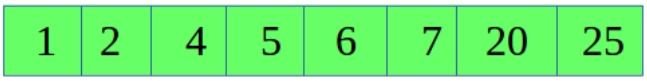
Maximum set bit sum in array without considering adjacent elements - GeeksforGeeks

Given an array of integers arr[]. The task is to find the maximum sum of set bits(of the array elements) without adding the set bits of adjacent elements of the array.

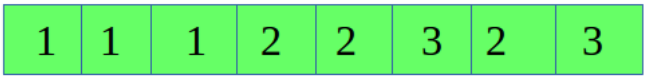
Examples:

```
Input : arr[] = {1, 2, 4, 5, 6, 7, 20, 25}
Output : 9

Input : arr[] = {5, 7, 9, 5, 13, 7, 20, 25}
Output : 11
```

Given an array of numbers



Total number of set bits for a particular number are



Maximum number of set bits without adding the adjacent bits are
:-1+2+3+3
max=9

Approach:

1. First of all, find the total number of set bits for every element of the array and store them in a different array or the same array(to avoid using extra space).
2. Now, the problem is reduced to find the maximum sum in the array such that no two elements are adjacent.
3. Loop for all elements in arr[] and maintain two sums incl and excl where incl = Max sum including the previous element and excl = Max sum excluding the previous element.
4. Max sum excluding the current element will be max(incl, excl) and max sum including the current element will be excl + current element (Note that only excl is considered because elements cannot be adjacent).
5. At the end of the loop return max of incl and excl.

Below is the implementation of the above approach:

C++

```
 #include<bits/stdc++.h>
using namespace std;

// Function to count total number
// of set bits in an integer
int bit(int n)
```

```
{
    int count = 0;

    while(n)
    {
        count++;
        n = n & (n - 1);
    }

    return count;
}

// Maximum sum of set bits
int maxSumOfBits(int arr[], int n)
{
    // Calculate total number of
    // set bits for every element
    // of the array
    for(int i = 0; i < n; i++)
    {
        // find total set bits for
        // each number and store
        // back into the array
        arr[i] = bit(arr[i]);
    }

    int incl = arr[0];
    int excl = 0;
    int excl_new;

    for (int i = 1; i < n; i++)
    {
        // current max excluding i
        excl_new = (incl > excl) ?
                            incl : excl;

        // current max including i
        incl = excl + arr[i];
        excl = excl_new;
    }

    // return max of incl and excl
    return ((incl > excl) ?
                     incl : excl);
}

// Driver code
int main()
```

```
{
    int arr[] = {1, 2, 4, 5,
                 6, 7, 20, 25};

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << maxSumOfBits(arr, n);

    return 0;
}
```

Java

```
 import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{
// Function to count total number
// of set bits in an integer
static int bit(int n)
{
    int count = 0;

    while(n > 0)
    {
        count++;
        n = n & (n - 1);
    }

    return count;
}

// Maximum sum of set bits
static int maxSumOfBits(int arr[], int n)
{
// Calculate total number of set bits
// for every element of the array
for(int i = 0; i < n; i++)
{
    // find total set bits for
    // each number and store
    // back into the array
    arr[i] = bit(arr[i]);
}

int incl = arr[0];
```

```
int excl = 0;
int excl_new;

for (int i = 1; i < n; i++)
{
    // current max excluding i
    excl_new = (incl > excl) ?
                        incl : excl;

    // current max including i
    incl = excl + arr[i];
    excl = excl_new;
}

// return max of incl and excl
return ((incl > excl) ?
                 incl : excl);
}

// Driver code
public static void main(String args[])
{
    int arr[] = {1, 2, 4, 5,
                 6, 7, 20, 25};

    int n = arr.length;

    System.out.print(maxSumOfBits(arr, n));
}
}

// This code is contributed
// by Subhadeep
```

C#

using System;

class GFG
{
// Function to count total number
// of set bits in an integer
static int bit(int n)
{
int count = 0;

while(n > 0)
{
count++;

```
n = n & (n – 1);
}

return count;
}

// Maximum sum of set bits
static int maxSumOfBits(int []arr, int n)
{

// Calculate total number of set bits
// for every element of the array
for(int i = 0; i < n; i++) { // find total set bits for // each number and store // back into
the array arr[i] = bit(arr[i]); } int incl = arr[0]; int excl = 0; int excl_new; for (int i = 1; i
< n; i++) { // current max excluding i excl_new = (incl > excl) ?
incl : excl;

// current max including i
incl = excl + arr[i];
excl = excl_new;
}

// return max of incl and excl
return ((incl > excl) ?
incl : excl);
}

// Driver code
public static void Main()
{
int []arr = {1, 2, 4, 5,
6, 7, 20, 25};

int n = arr.Length;

Console.WriteLine(maxSumOfBits(arr, n));
}
}

// This code is contributed
// by chandan_jnu.
```

Output:

```
9
```

Time Complexity: O(Nlogn)
Auxiliary Space: O(1)

Note: Above code can be optimised to **O(N)** using ___builtin_popcount function to count set bits in **O(1) time**.

Improved By : tufan_gupta2000, Chandan_Kumar

Source

https://www.geeksforgeeks.org/maximum-set-bit-sum-in-array-without-considering-adjacent-elements/

Chapter 214

Maximum steps to transform 0 to X with bitwise AND

Maximum steps to transform 0 to X with bitwise AND - GeeksforGeeks

For an integer N, there are elements ranging from 0 to N-1. Certain elements can be transformed to other elements. Each transformation requires a certain effort which is equal to 1 unit, for each transformation. An element A can be transformed to element B, if and only if A != B and A & B = A (where & is the bitwise AND operator). We need find the maximum effort possible to obtain the element with value X, from the element with value 0, by a series of transformations.

Examples :

> Input : X = 2
> Output : 1
> The only way of obtaining 2 is to directly transform 0 to 2 (bitwise AND of 0 and 2 is 0) and hence requires one step.
>
> Input : X = 3
> Output : 2
> 3 can be obtained in two steps. First, transform 0 to 1 (bitwise AND of 0 and 1 is 0). Then, transform 1 to 3 (bitwise AND of 1 and 3 is 1).

The simple solution is to count the number of set bits in X.
Explanation:
First, consider a **single step** transformation from A to B. All the set bits (bits which are equal to 1) of A should be set in B, otherwise bitwise AND of A and B will not be equal to A. If there is any bit which is set in A but not in B, then the transformation is not possible. The unset bits of A can either be set or unset in B, it does not matter. We can then change A to B by setting all the unset bits in A in one single step. Consequently, if we had to transform 0 to X in least steps, the answer would have been one because bitwise AND of 0 with any number is 0.
But we have to compute the maximum steps. So in each step, we set each bit starting from

the right and a set bit can not be cleared once it set.
Example:
Suppose we want to obtain 13 (1101 in binary) from 0. We start by setting the 1st bit from the right by transforming 0 to 1 (0001 in binary). We next set the 3rd bit from the right to form 5 (0101 in binary). The last step would be to set the 4th bit and obtain 13 (1101).

C++

```
 // CPP code to find the maximum possible
// effort
#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;
}

// Driver code
int main()
{
    int i = 3;
    cout << countSetBits(i);
    return 0;
}
```

Java

```
 // Java code to find the maximum
// possible effort

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;
}

// Driver code
public static void main(String[] args)
{
    int i = 3;
    System.out.print(countSetBits(i));
}
}

// This code is contributed by Smitha.
```

Python3

```
 # Python3 code to find the
# maximum possible effort

# 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

# Driver code
i = 3
print (countSetBits(i))

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# code to find the maximum
// possible effort
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;
}

// Driver code
public static void Main(String[] args)
{
    int i = 3;
    Console.Write(countSetBits(i));
}
}

// This code is contributed by Smitha.
```

PHP

```
 <?php
// PHP code to find the
// maximum possible effort

// 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;
}

// Driver code
```

```
$i = 3;
echo (countSetBits($i));

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output :

```
2
```

Improved By : Smitha Dinesh Semwal, manishshaw1

Source

https://www.geeksforgeeks.org/maximum-steps-transform-0-x-bitwise/

Chapter 215

Maximum subset with bitwise OR equal to k

Maximum subset with bitwise OR equal to k - GeeksforGeeks

Given an array of non negative integers and an integer k, find the subset of maximum length with bitwise OR equal to k.

Examples :

```
Input : arr[] = [1, 4, 2]
        k = 3
Output : [1, 2]
Explanation: The bitwise OR of
1 and 2 equals 3. It is not possible to obtain
a subset of length greater than 2.

Input : arr[] = [1, 2, 5]
        k = 4
Output : []
No subset's bitwise OR equals 4.
```

Method 1(Simple):
The naive method would be to consider all the subsets. While considering a subset, compute its bitwise OR. If it equals k, compare the subset's length with the maximum length so far and update the maximum length if required.

Method 2(Efficient):
0 OR 0 = 0
1 OR 0 = 1
1 OR 1 = 1
Hence, for all the positions in the binary representation of k with the bit equal to 0, the

corresponding position in the binary representations of all the elements in the resulting subset should necessarily be 0.
On the other hand, for positions in k with the bit equal to 1, there has to be at least one element with a 1 in the corresponding position. Rest of the elements can have either 0 or 1 in that position, it does not matter.

Therefore, to obtain the resulting subset, traverse the initial array. While deciding if the element should be in the resulting subset or not, check whether there is any position in the binary representation of k which is 0 and the corresponding position in that element is 1. If there exists such a position, then ignore that element, else include it in the resulting subset.

How to determine if there exists a position in the binary representation of k which is 0 and the corresponding position in an element is 1?
Simply take bitwise OR of k and that element. If it does not equal to k, then there exists such a position and the element has to be ignored. If their bitwise OR equals to k, then include the current element in the resulting subset.

The final step is to determine if there is at least one element with a 1 in a position with 1 in the corresponding position in k.
Simply compute the bitwise OR of the resulting subset. If it equals to k, then this is the final answer. Else no subset exists which satisfies the condition.

C++

```
 // CPP Program to find the maximum subset
// with bitwise OR equal to k
#include <bits/stdc++.h>
using namespace std;

// function to find the maximum subset with
// bitwise OR equal to k
void subsetBitwiseORk(int arr[], int n, int k)
{
    vector<int> v;

    for (int i = 0; i < n; i++) {

        // If the bitwise OR of k and element
        // is equal to k, then include that element
        // in the subset
        if ((arr[i] | k) == k)
            v.push_back(arr[i]);
    }

    // Store the bitwise OR of elements in v
    int ans = 0;

    for (int i = 0; i < v.size(); i++)
        ans |= v[i];
```

```
    // If ans is not equal to k, subset doesn't exist
    if (ans != k) {
        cout << "Subset does not exist" << endl;
        return;
    }

    for (int i = 0; i < v.size(); i++)
        cout << v[i] << ' ';
}

// Driver Code
int main()
{
    int k = 3;
    int arr[] = { 1, 4, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    subsetBitwiseORk(arr, n, k);
    return 0;
}
```

Java

```
 // Java Program to find the maximum subset
// with bitwise OR equal to k
import java.util.*;

class GFG {

    // function to find the maximum subset
    // with bitwise OR equal to k
    static void subsetBitwiseORk(int arr[],
                                 int n, int k)
    {
        ArrayList<Integer> v =
                  new ArrayList<Integer>();

        for (int i = 0; i < n; i++) {

            // If the bitwise OR of k and
            // element is equal to k, then
            // include that element in the
            // subset
            if ((arr[i] | k) == k){
                v.add(arr[i]);
            }
        }
```

```
        // Store the bitwise OR of elements
        // in v
        int ans = 0;

        for (int i = 0; i < v.size(); i++)
            ans = ans|v.get(i);

        // If ans is not equal to k, subset
        // doesn't exist
        if (ans != k) {
            System.out.println("Subset does"
                           + " not exist" );
            return;
        }

        for (int i = 0; i < v.size(); i++)
            System.out.print(v.get(i) + " " );
    }

    // main function
    public static void main(String[] args)
    {
        int k = 3;
        int arr[] = { 1, 4, 2 };
        int n = arr.length;

        subsetBitwiseORk(arr, n, k);

    }
}

// This code is contributed by Arnab Kundu.
```

Python3

```
 # Python3 Program to find the
# maximum subset with bitwise
# OR equal to k

# function to find the maximum
# subset with bitwise OR equal to k
def subsetBitwiseORk(arr, n, k) :
    v = []

    for i in range(0, n) :
        # If the bitwise OR of k
        # and element is equal to k,
```

```
        # then include that element
        # in the subset
        if ((arr[i] | k) == k) :
            v.append(arr[i])

    # Store the bitwise OR
    # of elements in v
    ans = 0

    for i in range(0, len(v)) :
        ans |= v[i]

    # If ans is not equal to
    # k, subset doesn't exist
    if (ans != k) :
        print ("Subset does not exist\n")
        return

    for i in range(0, len(v)) :
        print ("{} ".format(v[i]), end="")

# Driver Code
k = 3
arr = [1, 4, 2]
n = len(arr)

subsetBitwiseORk(arr, n, k)

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# Program to find the maximum subset
// with bitwise OR equal to k
using System;
using System.Collections;

class GFG {

    // function to find the maximum subset
    // with bitwise OR equal to k
    static void subsetBitwiseORk(int []arr,
                                 int n, int k)
    {
        ArrayList v = new ArrayList();

        for (int i = 0; i < n; i++) {
```

```
            // If the bitwise OR of k and
            // element is equal to k, then
            // include that element in the
            // subset
            if ((arr[i] | k) == k){
                v.Add(arr[i]);
            }
        }

        // Store the bitwise OR of
        // elements in v
        int ans = 0;

        for (int i = 0; i < v.Count; i++)
            ans = ans|(int)v[i];

        // If ans is not equal to k, subset
        // doesn't exist
        if (ans != k) {
            Console.WriteLine("Subset does"
                          + " not exist" );
            return;
        }

        for (int i = 0; i < v.Count; i++)
            Console.Write((int)v[i] + " " );
    }

    // main function
    static public void Main(String []args)
    {
        int k = 3;
        int []arr = { 1, 4, 2 };
        int n = arr.Length;

        subsetBitwiseORk(arr, n, k);

    }
}

// This code is contributed by Arnab Kundu
```

PHP

```
 <?php
// PHP Program to find the
// maximum subset with bitwise
```

```
// OR equal to k

// function to find the maximum
// subset with bitwise OR equal to k
function subsetBitwiseORk($arr, $n, $k)
{
    $v = array();

    for ($i = 0; $i < $n; $i++)
    {

        // If the bitwise OR of k
        // and element is equal to k,
        // then include that element
        // in the subset
        if (($arr[$i] | $k) == $k)
            array_push($v, $arr[$i]);
    }

    // Store the bitwise OR
    // of elements in v
    $ans = 0;

    for ($i = 0; $i < count($v); $i++)
        $ans |= $v[$i];

    // If ans is not equal to
    // k, subset doesn't exist
    if ($ans != $k)
    {
        echo ("Subset does not exist\n");
        return;
    }

    for ($i = 0; $i < count($v); $i++)
        echo ($v[$i]." ");
}

// Driver Code
$k = 3;
$arr = array(1, 4, 2);
$n = count($arr);

subsetBitwiseORk($arr, $n, $k);

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output :

```
1 2
```

Time complexity : O(N), where N is the size of array.

Improved By : andrew1234, manishshaw1

Source

https://www.geeksforgeeks.org/maximum-subset-bitwise-equal-k/

Chapter 216

Maximum sum by adding numbers with same number of set bits

Maximum sum by adding numbers with same number of set bits - GeeksforGeeks

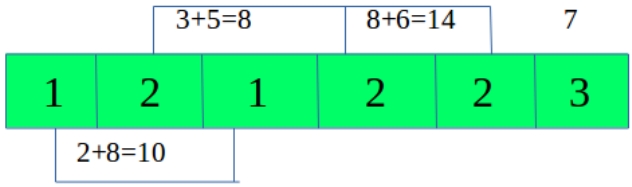
Given an array of N numbers, the task is to find the maximum sum that can be obtained by adding numbers with the same number of set bits.

Examples:

> **Input:** 32 3 7 5 27 28
> **Output:** 34
>
> **Input:** 2 3 8 5 6 7
> **Output:** 14

An array is given which contain N numbers

calculate the number of set bits for each number and store the result at the same position in another array .



So the maximum sum is equal 14.

Approach:

- Traverse in the array and count the number of set bits for every element.
- Initialize an array for 32 bits, assuming the number to have a maximum of 32 set bits.
- Iterate in the array and add the array element to the position which indicates the number of set bits.
- Traverse and find the maximum sum and return it.

Below is the implementation of the above approach:

C++

```
 // C++ program to find maximum sum
// by adding numbers with same number of set bits
#include <bits/stdc++.h>
using namespace std;

// count the number of bits
// for each element of array
int bit_count(int n)
{
    int count = 0;

    // Count the number of set bits
    while (n) {
        count++;
```

```
        n = n & (n - 1);
    }

    return count;
}

// Function to return the
// the maximum sum
int maxsum(int arr[], int n)
{
    int bits[n];

    // Calculate the
    for (int i = 0; i < n; i++) {
        bits[i] = bit_count(arr[i]);
    }

    // Assuming the number to be
    // a maximum of 32 bits
    int sum[32] = { 0 };

    // Add the number to the
    // number of set bits
    for (int i = 0; i < n; i++) {
        sum[bits[i]] += arr[i];
    }

    int maximum = 0;

    // Find the maximum sum
    for (int i = 0; i < 32; i++) {

        maximum = max(sum[i], maximum);
    }

    return maximum;
}

// Driver code
int main()
{

    int arr[] = { 2, 3, 8, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << maxsum(arr, n);

    return 0;
}
```

Java

```
 // Java program to find maximum sum
// by adding numbers with same number of set bits

class GFG
{
// count the number of bits
// for each element of array
static int bit_count(int n)
{
    int count = 0;

    // Count the number of set bits
    while (n>0)
    {
        count++;

        n = n & (n - 1);
    }

    return count;
}

// Function to return the
// the maximum sum
static int maxsum(int[] arr, int n)
{
    int[] bits=new int[n];

    // Calculate the
    for (int i = 0; i < n; i++)
    {
        bits[i] = bit_count(arr[i]);
    }

    // Assuming the number to be
    // a maximum of 32 bits
    int[] sum=new int[32];

    // Add the number to the
    // number of set bits
    for (int i = 0; i < n; i++)
    {
        sum[bits[i]] += arr[i];
    }

    int maximum = 0;
```

```
    // Find the maximum sum
    for (int i = 0; i < 32; i++)
    {

        maximum = Math.max(sum[i], maximum);
    }

    return maximum;
}

// Driver code
public static void main (String[] args)
{
    int[] arr = { 2 ,3 , 8, 5, 6, 7 };
    int n = arr.length;
    System.out.println(maxsum(arr, n));

}
}

// This Code is contributed by mits
```

C#

```
 // C# program to find maximum
// sum by adding numbers with
// same number of set bits
using System;

class GFG
{
// count the number of bits
// for each element of array
static int bit_count(int n)
{
    int count = 0;

    // Count the number
    // of set bits
    while (n > 0)
    {
        count++;

        n = n & (n - 1);
    }

    return count;
```

```
}

// Function to return the
// the maximum sum
static int maxsum(int[] arr, int n)
{
    int[] bits = new int[n];

    // Calculate the
    for (int i = 0; i < n; i++)
    {
        bits[i] = bit_count(arr[i]);
    }

    // Assuming the number to be
    // a maximum of 32 bits
    int[] sum = new int[32];

    // Add the number to the
    // number of set bits
    for (int i = 0; i < n; i++)
    {
        sum[bits[i]] += arr[i];
    }

    int maximum = 0;

    // Find the maximum sum
    for (int i = 0; i < 32; i++)
    {
        maximum = Math.Max(sum[i], maximum);
    }

    return maximum;
}

// Driver code
static void Main()
{
    int[] arr = { 2 ,3 , 8, 5, 6, 7 };
    int n = arr.Length;
    Console.WriteLine(maxsum(arr, n));
}
}

// This Code is contributed by mits
```

PHP

Output:

```
14
```

Time Complexity: O(N * 32)
Auxiliary Space: O(N)

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/maximum-sum-by-adding-numbers-with-same-number-of-set-bits/

Chapter 217

Minimum bit changes in Binary Circular array to reach a index

Minimum bit changes in Binary Circular array to reach a index - GeeksforGeeks

Given a Binary Circular Array of size **N** elements and two positive integers **x** and **y** indicating the indices in the circular array. The task is check which path, clockwise or anti-clockwise, from index x to index y, we face the minimum number bit flips. Output "Clockwise" or "Anti-clockwise" and the value of minimum bit flip, in case of equal count output "Clockwise".

Examples:

```
Input : arr[] = { 0, 0, 0, 1, 1, 0 }
        x = 0, y = 5
Output : Anti-clockwise 0
The path 0 -> 1 -> 2 -> 3 -> 4 -> 5, we have only 1 value change i.e from index 2 to 3.
The path 0 -> 5 have 0 value change.
So, the answer is Anti-clockwise 0.

Input : s = { 1, 1, 0, 1, 1 }
        x = 2, y = 0
Output : Clockwise 1
```

The idea is to check by going once Clockwise and store the count1 and then going anti-clockwise and store the count2. Then output by comparing count1 and count2.

How to travel clockwise or anticlockwise?
It will be hard to travel clockwise in the array where x > y and same in case of anticlockwise where y > x. So, we will store the given binary array in the string "S". And to make it circular, we will append S to S i.e S = S + S. We will make the adjustment in x and y to travel clockwise or anticlockwise.

Now, if y > x and to go clockwise, it will be easy to iterate from x to y and calculate the number of flip bits.
If y > x and to go anti-clockwise, we will add |S| to x then iterate from y to x and calculate the number of flip bits.

Now, if x > y, we will swap x and y and calculate the answer using above approach. Then output the opposite of the result .

To calculate the number of flip bits, just store the current bit of index and check if next index have the same bit as current. If yes then do nothing else change the current bit to the bit of the next index and increment minimum bit by 1.

Below is the C++ implementation of this approach:

```
 // CPP program to find direction with minimum flips
#include <bits/stdc++.h>
using namespace std;

// finding which path have minimum flip bit and
// the minimum flip bits
void minimumFlip(string s, int x, int y)
{
    // concatenating given strin to itself,
    // to make it circular
    s = s + s;

    // check x is greater than y.
    // marking if output need to
    // be opposite.
    bool isOpposite = false;
    if (x > y) {
        swap(x, y);
        isOpposite = true;
    }

    // iterate Clockwise
    int valClockwise = 0;
    char cur = s[x];
    for (int i = x; i <= y; i++) {

        // if current bit is not equal
        // to next index bit.
        if (s[i] != cur) {
            cur = s[i];
            valClockwise++;
        }
    }

    // iterate Anti-Clockwise
```

```
    int valAnticlockwise = 0;
    cur = s[y];
    x += s.length();
    for (int i = y; i <= x; i++) {

        // if current bit is not equal
        // to next index bit.
        if (s[i] != cur) {
            cur = s[i];
            valAnticlockwise++;
        }
    }

    // Finding whether Clockwise or Anti-clockwise
    // path take minimum flip.
    if (valClockwise <= valAnticlockwise) {
        if (!isOpposite)
            cout << "Clockwise "
                << valClockwise << endl;
        else
            cout << "Anti-clockwise "
                 << valAnticlockwise << endl;
    }
    else {
        if (!isOpposite)
            cout << "Anti-clockwise "
                 << valAnticlockwise << endl;
        else
            cout << "Clockwise "
                 << valClockwise << endl;
    }
}

// Driven Program
int main()
{
    int x = 0, y = 8;
    string s = "000110";
    minimumFlip(s, x, y);
    return 0;
}
```

Output

```
Anti-clockwise 0
```

Source

https://www.geeksforgeeks.org/minimum-bit-changes-binary-circular-array-reach-index/

Chapter 218

Minimum bitwise operations to convert given a into b.

Minimum bitwise operations to convert given a into b. - GeeksforGeeks

Given two positive integer a and b you have to change a to b by applying any of the three operations on binary form of a. You can select ai and aj (any two bits where i!=j) from binary form of a and then perform operation as:

- AND operation as : temp = ai & aj, ai = temp & ai, aj = temp & aj
- OR operation as : temp = ai | aj, ai = temp | ai, aj = temp | aj
- XOR operation as : temp = ai ^ aj, ai = temp ^ ai, aj = temp ^ aj

where & = bitwise AND, | = bitwiese OR and ^ = bitwise XOR.
Find the minimum operation required for conversion of a to b. Also, if conversion of a to b is not possible then print -1.

Examples:

```
Input : a = 12 (1100), b = 10 (1010)
Output : 1
Explanation : select a2 and a3 and perform XOR

Input : a = 15 (1111), b = 10 (1010)
Output : -1
Explanation : Conversion from a to b is not possible
```

Explanation : First of all let's understand the working of all three operation.

1. **AND operation** as : temp = ai & aj, ai = temp & ai, aj = temp & aj
2. **OR operation** as : temp = ai | aj, ai = temp | ai, aj = temp | aj

3. **XOR operation** as : temp = ai ^ aj, ai = temp ^ ai, aj = temp ^ aj

Some conclusion on basis of working of operations :

1. If all bits of a are 1 or 0 then we can not change value of a.
2. If a equals to b then no operation required.
3. Let n be number of indices i, where ai = 0 and bi = 1.
   Let m be number of indices i, where ai = 1 and bi = 0.

   Let us think about the n elements, where ai = 0 and bi = 1. We have to change all of these zeros into ones. Note that this will require at least n operations.
   Similarly for all the m elements, where ai = 1 and bi = 0. We have to change all of these ones into zeros. Note that this will require at least m operations.

   Let res = max(n, m). We can make the a an b equal in res operations as follows.

   Let n >= m. Take m 1's and n 0's in A and apply the XOR operation to swap 0's with 1's. After that you will be left with total n-m zeros elements to change to one. That you can do by taking each of these zeros with some single one and applying the OR operation.
   Let m >= n. Take m 1's and n 0's in A and apply the XOR operation to swap 0's with 1's. After that you will be left with total m-n ones elements to change to zero. That you can do by taking each of these ones with some single zero and applying the OR operation.

```
 // Cpp program to find min operation to convert a to b
#include <bits/stdc++.h>
using namespace std;

// function to return min operation
int minOp(bitset<32> a1, bitset<32> b1)
{
    // if a1 == b1 return 0
    if (a1 == b1)
        return 0;

    // if all bits of a = 0
    if (a1 == 0)
        return -1;

    // if all bits of a =1
    // first convert a1 to int and then cal a1 & a1+1
    if (((int)a1.to_ulong() & ((int)a1.to_ulong() + 1))
                                                == 0)
        return -1;

    // convert a and b to binary string
    string a = a1.to_string();
```

```
    string b = b1.to_string();

    // check where ai and bi are different
    // and count n where ai = 1 and m where ai = 0
    int n = 0, m = 0;
    for (int i = 0; i < b.size(); i++) {
        if (b[i] != a[i]) {
            if (a[i] == '1')
                n++;
            else
                m++;
        }
    }

    // return result
    return max(n, m);
}

// driver program
int main()
{
    bitset<32> a = 14, b = 1;
    cout << minOp(a, b);
    return 0;
}
```

Output:

```
3
```

Source

https://www.geeksforgeeks.org/minimum-bitwise-operations-to-convert-given-a-into-b/

Chapter 219

Minimum digits to remove to make a number Perfect Square

Minimum digits to remove to make a number Perfect Square - GeeksforGeeks

Given a integer n, we need to find how many digits remove from the number to make it a perfect square.

Examples :

> Input : 8314
> Output: 81 2
> Explanation: If we remove 3 and 4 number becomes 81 which is a perfect square.
>
> Input : 57
> Output : -1

The idea is to generate all possible subsequences and return optimal string using set bits. Let's suppose we have a string 8314. And using set bits we form all possible subsequences i.e.,

8, 3, 83, 1, 81, 31, 831, 4, 84, 34, 834, 14, 814, 314, 8314.

After forming all possible subsequences, we check which one is the perfect square. And we return a perfect square number which has the minimum length.

In above example, three perfect squares are 1 4 and 81, so answer would be 81 because 81 has the max length 2.

C++

```
 // C++ program to find required minimum digits
// need to remove to make a number perfect squre
#include <bits/stdc++.h>
```

```
using namespace std;

// function to check minimum number of digits
// should be removed to make this number
// a perfect square
int perfectSquare(string s)
{
    // size of the string
    int n = s.size();

    // our final answer
    int ans = -1;

    // to store string which is perfect square.
    string num;

    // We make all possible subsequences
    for (int i = 1; i < (1 << n); i++) {
        string str = "";

        for (int j = 0; j < n; j++) {

            // to check jth bit is set or not.
            if ((i >> j) & 1) {
                str += s[j];
            }
        }

        // we do not consider a number with leading zeros
        if (str[0] != '0') {

            // convert our temporary string into integer
            int temp = 0;
            for (int j = 0; j < str.size(); j++)
                temp = temp * 10 + (int)(str[j] - '0');

            int k = sqrt(temp);

            // checking temp is perfect square or not.
            if (k * k == temp) {

                // taking maximum sized string
                if (ans < (int)str.size()) {
                    ans = (int)str.size();
                    num = str;
                }
            }
        }
```

```
    }

    if (ans == -1)
        return ans;
    else {

        // print PerfectSquare
        cout << num << " ";
        return n - ans;
    }
}

// Driver code
int main()
{
    cout << perfectSquare("8314") << endl;
    cout << perfectSquare("753") << endl;
    return 0;
}
```

Java

```
 // Java program to find required minimum digits
// need to remove to make a number perfect squre
import java.io.*;
import java.lang.*;

public class GFG {

    // function to check minimum
    // number of digits should
    // be removed to make this
    // number a perfect square
    static int perfectSquare(String s)
    {
        // size of the string
        int n = s.length();

        // our final answer
        int ans = -1;

        // to store string which
        // is perfect square.
        String num = "";

        // We make all possible subsequences
        for (int i = 1; i < (1 << n); i++) {
            String str = "";
```

```
        for (int j = 0; j < n; j++) {

            // to check jth bit is set or not.
            if (((i >> j) & 1) == 1) {
                str += s.charAt(j);
            }
        }

        // we do not consider a number
        // with leading zeros
        if (str.charAt(0) != '0') {

            // convert our temporary
            // string into integer
            int temp = 0;
            for (int j = 0; j <
                         str.length(); j++)
                temp = temp * 10 +
                  (int)(str.charAt(j) - '0');

            int k = (int)Math.sqrt(temp);

            // checking temp is perfect
            // square or not.
            if (k * k == temp) {

                // taking maximum sized string
                if (ans < (int)str.length()) {
                    ans = (int)str.length();
                    num = str;
                }
            }
        }
    }

    if (ans == -1)
        return ans;
    else {

        // print PerfectSquare
        System.out.print(num + " ");
        return n - ans;
    }
}

// Driver code
public static void main(String args[])
```

```
    {
        System.out.println(perfectSquare("8314"));
        System.out.println(perfectSquare("753"));
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

Python3

```
 # C++ program to find required minimum
# digits need to remove to make a
# number perfect squre

import math
# function to check minimum number of
# digits should be removed to make
# this number a perfect square
def perfectSquare(s) :

    # size of the string
    n = len(s)

    # our final answer
    ans = -1

    # to store string which is
    # perfect square.
    num = ""

    # We make all possible subsequences
    for i in range(1, (1 << n)) :
        str = ""

        for j in range(0, n) :

            # to check jth bit is
            # set or not.
            if ((i >> j) & 1) :
                str = str + s[j]

        # we do not consider a number
        # with leading zeros
        if (str[0] != '0') :

            # convert our temporary
            # string into integer
```

```
            temp = 0;
            for j in range(0, len(str)) :
                temp = (temp * 10 +
                 (ord(str[j]) - ord('0')))

            k = int(math.sqrt(temp))

            # checking temp is perfect
            # square or not.
            if (k * k == temp) :

                # taking maximum sized
                # string
                if (ans < len(str)) :
                    ans = len(str)
                    num = str

    if (ans == -1) :
        return ans
    else :

        # print PerfectSquare
        print ("{} ".format(num), end="")
        return n - ans

# Driver code
print (perfectSquare("8314"))
print (perfectSquare("753"));

# This code is contributed by
# manishshaw1.
```

C#

```
 // C# program to find required minimum digits
// need to remove to make a number perfect squre
using System;
class GFG {

    // function to check minimum
    // number of digits should
    // be removed to make this
    // number a perfect square
    static int perfectSquare(string s)
    {
        // size of the string
        int n = s.Length;
```

```
// our final answer
int ans = -1;

// to store string which
// is perfect square.
string num = "";

// We make all possible subsequences
for (int i = 1; i < (1 << n); i++) {
    string str = "";

    for (int j = 0; j < n; j++) {

        // to check jth bit is set or not.
        if (((i >> j) & 1) == 1) {
            str += s[j];
        }
    }

    // we do not consider a number
    // with leading zeros
    if (str[0] != '0') {

        // convert our temporary
        // string into integer
        int temp = 0;
        for (int j = 0; j < str.Length; j++)
            temp = temp * 10 + (int)(str[j] - '0');

        int k = (int)Math.Sqrt(temp);

        // checking temp is perfect
        // square or not.
        if (k * k == temp) {

            // taking maximum sized string
            if (ans < (int)str.Length) {
                ans = (int)str.Length;
                num = str;
            }
        }
    }
}

if (ans == -1)
    return ans;
else {
```

```
            // print PerfectSquare
            Console.Write(num + " ");
            return n - ans;
        }
    }

    // Driver code
    public static void Main()
    {
        Console.WriteLine(perfectSquare("8314"));
        Console.WriteLine(perfectSquare("753"));
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

PHP

```
 <?php
// PHP program to find required
// minimum digits need to remove
// to make a number perfect squre

// function to check minimum
// number of digits should be
// removed to make this number
// a perfect square
function perfectSquare($s)
{
    // size of the string
    $n = strlen($s);

    // our final answer
    $ans = -1;

    // to store string which
    // is perfect square.
    $num = "";

    // We make all possible
    // subsequences
    for ($i = 1; $i < (1 << $n); $i++)
    {
        $str = "";
        for ($j = 0; $j < $n; $j++)
        {
```

```
            // to check jth bit
            // is set or not.
            if (($i >> $j) & 1)
            {
                $str = $str.$s[$j];
            }
        }

        // we do not consider a
        // number with leading zeros
        if ($str[0] != '0')
        {
            // convert our temporary
            // string into integer
            $temp = 0;
            for ($j = 0; $j < strlen($str); $j++)
                $temp = $temp * 10 +
                        (ord($str[$j]) - ord('0'));

            $k = (int)(sqrt($temp));

            // checking temp is perfect
            // square or not.
            if (($k * $k) == $temp)
            {

                // taking maximum sized string
                if ($ans < strlen($str))
                {
                    $ans = strlen($str);
                    $num = $str;
                }
            }
        }
    }

    if ($ans == -1)
        return $ans;
    else
    {
        // print PerfectSquare
        echo ($num." ");
        return ($n - $ans);
    }
}

// Driver code
echo (perfectSquare("8314"). "\n");
```

```
echo (perfectSquare("753"). "\n");

// This code is contributed by
// Manish Shaw (manishshaw1)
?>
```

Output :

```
81 2
-1
```

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/required-minimum-digits-remove-number-make-perfect-square/

Chapter 220

Minimum flips required to maximize a number with k set bits

Minimum flips required to maximize a number with k set bits - GeeksforGeeks

Given two numbers n and k, we need to find the minimum number of flips required to maximize given number by flipping its bits such that the resulting number has exactly k set bits.
Note : K must be less than number of bits in n.

Examples :

```
Input : n = 14, k = 2
Output : Min Flips = 1
Explanation :
Binary representation of 14 = 1110
Largest 4-digit Binary number with
2 set bit = 1100
Conversion from 1110 to 1100
requires 1 flipping

Input : n = 145, k = 4
Output : Min Flips = 3
Explanation :
Binary representation of 145 = 10010001
Largest 8-digit Binary number with
4 set bit = 11110000
Conversion from 10010001 to 11110000
requires 3 flipping
```

For the given number n and k find the largest number possible with k-set bits and having exactly same number of bits as n has as :

- size = log2(n) + 1 gives the number of bits of n.
- max = pow(2, k) – 1 gives largest possible number with k bits.
- max = max << (size – k) gives the largest number possible with k-set bits and having exactly same number of bits as n has
- Number of set bit in (n XOR max) is our required number of flipping.

Illustration of above aproach :

```
let n = 145 (10010001), k = 4

size = log2(n) + 1 = log2(145) + 1
                   = 7 + 1 = 8

max = pow(2, k) -1 = pow(2, 4) - 1
                   = 16 - 1 = 15 (1111)

max = max << (size - k) = 15 << (8 - 4)
                        = 240 (11110000)

number of set bit in  =  no. of set bit in
(n XOR max )              (145 ^ 240 )
                      = 3
```

C++

```
 // CPP for finding min flip
// for maximizing given n
#include <bits/stdc++.h>
using namespace std;

// function for finding set bit
int setBit(int xorValue)
{
    int count = 0;
    while (xorValue) {
        if (xorValue % 2)
            count++;

        xorValue /= 2;
    }

    // return count of set bit
    return count;
}
```

```
// function for finding min flip
int minFlip(int n, int k)
{
    // number of bits in n
    int size = log2(n) + 1;

    // Find the largest number of
    // same size with k set bits
    int max = pow(2, k) - 1;
    max = max << (size - k);

    // Count bit differences to find
    // required flipping.
    int xorValue = (n ^ max);
    return (setBit(xorValue));
}

// driver program
int main()
{
    int n = 27, k = 3;
    cout << "Min Flips = " << minFlip(n, k);
    return 0;
}
```

Java

```
 // JAVA Code to find Minimum flips required
// to maximize a number with k set bits
import java.util.*;

class GFG {

    // function for finding set bit
    static int setBit(int xorValue)
    {
        int count = 0;
        while (xorValue >= 1) {
            if (xorValue % 2 == 1)
                count++;

            xorValue /= 2;
        }

        // return count of set bit
        return count;
    }
```

```
    // function for finding min flip
    static int minFlip(int n, int k)
    {
        // number of bits in n
        int size = (int)(Math.log(n) /
                         Math.log(2)) + 1;

        // Find the largest number of
        // same size with k set bits
        int max = (int)Math.pow(2, k) - 1;
        max = max << (size - k);

        // Count bit differences to find
        // required flipping.
        int xorValue = (n ^ max);
        return (setBit(xorValue));
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
         int n = 27, k = 3;
         System.out.println("Min Flips = "+
                            minFlip(n, k));
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python 3

```
 # Python3 for finding min flip
# for maximizing given n
import math

# function for finding set bit
def setBit(xorValue):

    count = 0
    while (xorValue):
        if (xorValue % 2):
            count += 1

        xorValue = int(xorValue / 2)

    # return count
    # of set bit
```

```
    return count

# function for
# finding min flip
def minFlip(n, k):

    # number of bits in n
    size = int(math.log(n) /
               math.log(2) + 1)

    # Find the largest number of
    # same size with k set bits
    max = pow(2, k) - 1
    max = max << (size - k)

    # Count bit differences to
    # find required flipping.
    xorValue = (n ^ max)
    return (setBit(xorValue))

# Driver Code
n = 27
k = 3
print("Min Flips = " ,
        minFlip(n, k))

# This code is contributed
# by Smitha
```

C#

```
 // C# Code to find Minimum flips required
// to maximize a number with k set bits
using System;

class GFG {

    // function for finding set bit
    static int setBit(int xorValue)
    {
        int count = 0;
        while (xorValue >= 1) {
            if (xorValue % 2 == 1)
                count++;

            xorValue /= 2;
        }
```

```
        // return count of set bit
        return count;
    }

    // function for finding min flip
    static int minFlip(int n, int k)
    {
        // number of bits in n
        int size = (int)(Math.Log(n) /
                         Math.Log(2)) + 1;

        // Find the largest number of
        // same size with k set bits
        int max = (int)Math.Pow(2, k) - 1;
        max = max << (size - k);

        // Count bit differences to find
        // required flipping.
        int xorValue = (n ^ max);
        return (setBit(xorValue));
    }

    // Driver Code
    public static void Main()
    {
        int n = 27, k = 3;
        Console.Write("Min Flips = "+ minFlip(n, k));
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP for finding min flip
// for maximizing given n

// function for finding set bit
function setBit($xorValue)
{
    $count = 0;
    while ($xorValue)
    {
        if ($xorValue % 2)
            $count++;

        $xorValue /= 2;
```

```
    }

    // return count of set bit
    return $count;
}

// function for finding min flip
function minFlip($n, $k)
{
    // number of bits in n
    $size = log($n) + 1;

    // Find the largest number of
    // same size with k set bits
    $max = pow(2, $k) - 1;
    $max = $max << ($size - $k);

    // Count bit differences to find
    // required flipping.
    $xorValue = ($n ^ $max);
    return (setBit($xorValue));
}

// Driver Code
$n = 27; $k = 3;
echo "Min Flips = " , minFlip($n, $k);

// This code is contributed by vt_m.
?>
```

Output :

```
Min Flips = 3
```

Improved By : nitin mittal, vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/minimum-flips-required-to-maximize-a-number-with-k-set-bits/

Chapter 221

Minimum flips to make all 1s in left and 0s in right | Set 1 (Using Bitmask)

Minimum flips to make all 1s in left and 0s in right | Set 1 (Using Bitmask) - GeeksforGeeks

Given a binary array, we can flip all the 1 are in the left part and all the 0 to the right part.Calculate the minimum flips required to make all 1s in left and all 0s in right.

Examples:

```
Input: 1011000
Output: 1
1 flip is required to make it 1111000.

Input : 00001
Output : 2
2 flips required to make it 10000.
```

For solving this problem we use bitmasking. First, we convert this array to string, then we find the equivalent decimal number of that binary string. We try all masks with all possibilities of 1s in left and 0s in right. We iterate a loop till decimal number becomes zero. Each time we will do bitwise XOR of the number with mask and number of ones in XOR value will be the number of flips required. We decrease n by 1 and update the mask.

1-Take binary array as input
2-Convert array to string and then equivalent decimal number(num)
3-Take initial mask value and iterate till num <= 0
4-Find required flips using (num XOR mask)
5-Find minimum flips and decrease num and update mask
6-Return the minimum count

```
 // Java program to find minimum flips to make
// all 1s in left
import java.io.*;

class GFG {

    // function to count minimum number of flips
    public static int findMiniFlip(int[] nums)
    {
        int n = nums.length;
        String s = "";
        for (int i = 0; i < n; i++)
            s += nums[i];

        // This is converting string s into integer
        // of base 2 (if s = '100' then num = 4)
        long num = Integer.parseInt(s, 2);

        // initialize minXor with n that can be maximum
        // number of flips
        int minXor = n;

        // right shift 1 by (n-1) bits
        long mask = (1 << (n-1));
        while (n-1 > 0) {

            // calculate bitwise XOR of num and mask
            long temp = (num ^ mask);

            // Math.min(a, b) returns minimum of a and b
            // return minimum number of flips till that
            // digit
            minXor = Math.min(minXor, countones(temp));
            n--;

            mask = (mask | (1 << n));
        }
        return minXor;
    }

    // function to count number of 1s
    public static int countones(long n)
    {
        int c = 0;
        while (n > 0) {
            n = n & (n-1);
            c++;
        }
```

```
        return c;
    }

    public static void main(String[] args)
    {
        int[] nums = { 1, 0, 1, 1, 0, 0, 0 };
        int n = findMiniFlip(nums);
        System.out.println(n);
    }
}
```

Output:

```
1
```

Source

https://www.geeksforgeeks.org/minimum-flips-make-1s-left-0s-right-set-1-using-bitmask/

Chapter 222

Minimum number using set bits of a given number

Minimum number using set bits of a given number - GeeksforGeeks

Given an unsigned number, find the minimum number that could be formed by using the bits of the given unsigned number.

Examples :

> Input : 6
> Output : 3
> Binary representation of 6 is 0000….0110. Smallest number with same number of set bits 0000….0011.
>
> Input : 11
> Output : 7

Simple Approach:
1. Find binary representation of the number using simple decimal to binary representation technique.
2. Count number of set bits in the binary representation equal to 'n'.
3. Create a binary representation with it's 'n' least significant bits set to 1.
4. Convert the binary representation back to the number.

Efficient Approach:
1. Just measure the number of 1's present in the bit representation of the number.
2. (Number of set bits raised to the power of 2) – 1 represents the minimized number.

C++

```
 // An efficient C++ program to find
// minimum number formed by bits of a given number.
#include <bits/stdc++.h>
```

```
#define ll unsigned int
using namespace std;

// Returns minimum number formed by
// bits of a given number.
ll minimize(ll a)
{
    // _popcnt32(a) gives number of 1's
    // present in binary representation
    // of a.
    ll n = _popcnt32(a);

    return (pow(2, n) - 1);
}

// Driver function.
int main()
{
    ll a = 11;
    cout << minimize(a) << endl;
    return 0;
}
```

Java

```
 // An efficient Java program to
// find minimum number formed
// by bits of a given number.
import java.io.*;

class GFG
{
    public static int _popcnt32(long number)
    {
        int count = 0;
        while (number > 0)
        {
            count += number & 1L;
            number >>= 1L;
        }
        return count;
    }

    // Returns minimum number formed
    // by bits of a given number.
    static long minimize(long a)
    {
        // _popcnt32(a) gives number
```

```
        // of 1's present in binary
        // representation of a.
        int n = _popcnt32(a);

        return ((long)Math.pow(2, n) - 1);
    }

    // Driver Code.
    public static void main(String args[])
    {
        long a = 11;
        System.out.print(minimize(a));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

C#

```
 // An efficient C# program to
// find minimum number formed
// by bits of a given number.
using System;
using System.Linq;
using System.Collections.Generic;

class GFG
{
    // Returns minimum number formed
    // by bits of a given number.
    static long minimize(long a)
    {
        // _popcnt32(a) gives number
        // of 1's present in binary
        // representation of a.
        string binaryString = Convert.ToString(a, 2);
        int n = binaryString.Split(new [] {'0'},
                StringSplitOptions.RemoveEmptyEntries).Length + 1;

        return ((long)Math.Pow(2, n) - 1);
    }

    // Driver Code.
    static void Main()
    {
        long a = 11;
        Console.Write(minimize(a));
```

```
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output :

```
7
```

Note : The above code uses GCC specific functions. If we wish to write code for other compilers, we may use Count set bits in an integer.

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/minimum-number-using-set-bits-given-number/

Chapter 223

Minimum value of N such that xor from 1 to N is equal to K

Minimum value of N such that xor from 1 to N is equal to K - GeeksforGeeks

Given a value K which is the XOR of all the values from 1 to N, the task is to find the minimum value of N such that XOR from 1 to N is equal to K.

Examples:

```
Input: K = 7
Output: 6
1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 = 7

Input: K = 10
Output: Not Possible
```

Approach: This problem is similar to the Calculate XOR from 1 to n. Below are the conditions to be checked:

1. If k = 0, then N = 3.
2. If k = 1, then N = 1.
3. If k % 4 = 0, then N = k.
4. If k % 4 = 3, then N = k-1.

Below is the implementation of above approach:

C++

```
// C++ implementation of above approach
```

```
#include <bits/stdc++.h>
using namespace std;

// Function to find the value of N
int findN(int k)
{

    // variable to store the result
    int ans;

    // handling case for '0'
    if (k == 0)
        ans = 3;

    // handling case for '1'
    if (k == 1)
        ans = 1;

    // when number is completely divided by
    // 4 then minimum 'x' will be 'k'
    else if (k % 4 == 0)
        ans = k;

    // when number divided by 4 gives 3 as
    // remainder then minimum 'x' will be 'k-1'
    else if (k % 4 == 3)
        ans = k - 1;

    // else it is not possible to get
    // k for any value of x
    else
        ans = -1;

    return ans;
}

// Driver code
int main()
{
    // let the given number be 7
    int k = 7;

    int res = findN(k);
    if (res == -1)
        cout << "Not possible";
    else
        cout << res;
```

```
    return 0;
}
```

Java

```
 // Java implementation of
// above approach
import java.io.*;

class GFG
{

// Function to find the
// value of N
static int findN(int k)
{

    // variable to store
    // the result
    int ans;

    // handling case for '0'
    if (k == 0)
        ans = 3;

    // handling case for '1'
    if (k == 1)
        ans = 1;

    // when number is completely
    // divided by 4 then minimum
    // 'x' will be 'k'
    else if (k % 4 == 0)
        ans = k;

    // when number divided by 4
    // gives 3 as remainder then
    // minimum 'x' will be 'k-1'
    else if (k % 4 == 3)
        ans = k - 1;

    // else it is not possible to
    // get k for any value of x
    else
        ans = -1;

    return ans;
}
```

```
// Driver code
public static void main (String[] args)
{
    // let the given number be 7
    int k = 7;

    int res = findN(k);
    if (res == -1)
        System.out.println("Not possible");
    else
        System.out.println(res);
}
}

// This code is contributed
// by inder_verma
```

C#

```
// C# implementation of
// above approach
using System;

class GFG
{

// Function to find the
// value of N
static int findN(int k)
{

// variable to store
// the result
int ans;

// handling case for '0'
if (k == 0)
ans = 3;

// handling case for '1'
if (k == 1)
ans = 1;

// when number is completely
// divided by 4 then minimum
// 'x' will be 'k'
else if (k % 4 == 0)
ans = k;

// when number divided by 4
// gives 3 as remainder then
```

```
// minimum 'x' will be 'k-1'
else if (k % 4 == 3)
ans = k – 1;

// else it is not possible to
// get k for any value of x
else
ans = -1;

return ans;
}

// Driver code
public static void Main ()
{
// let the given number be 7
int k = 7;

int res = findN(k);
if (res == -1)
Console.WriteLine("Not possible");
else
Console.WriteLine(res);
}
}

// This code is contributed
// by inder_verma
```

Output:

```
6
```

How does this work?
When we do XOR of numbers, we get 0 as XOR value just before a multiple of 4. This keeps repeating before every multiple of 4.

```
Number Binary-Repr  XOR-from-1-to-n
1         1           [0001]
2        10           [0011]
3        11           [0000]  <----- We get a 0
4       100           [0100]  <----- Equals to n
5       101           [0001]
6       110           [0111]
7       111           [0000]  <----- We get 0
8      1000           [1000]  <----- Equals to n
9      1001           [0001]
10     1010           [1011]
```

```
11      1011            [0000] <------ We get 0
12      1100            [1100] <------ Equals to n
```

Improved By : inderDuMCA

Source

https://www.geeksforgeeks.org/minimum-value-of-n-such-that-xor-from-1-to-n-is-equal-to-k/

Chapter 224

Modify a bit at a given position

Modify a bit at a given position - GeeksforGeeks

Given a number **n**, a position **p** and a binary value **b**, we need to change the bit at position p in n to value b.

Examples :

```
Input : n = 7, p = 2, b = 0
Output : 3
7 is 00000111 after clearing bit at
2rd position, it becomes 0000011.

Input : n = 7, p = 3, b = 1
Output : 15
7 is 00000111 after setting bit at
3rd position it becomes 00001111.

We first create a mask that has set bit only
at given position using bit wise shift.
      mask = 1 << position

Then to change value of bit to b, we first
make it 0 using below operation
      value & ~mask

After changing it 0, we change it to b by
doing or of above expression with following
(b << p) & mask, i.e., we return
      (n & ~mask) | ((b << p) & mask)
```

Below is the implementation of above steps :

C++

```
 // CPP program to modify a bit at position
// p in n to b.
#include <bits/stdc++.h>
using namespace std;

// Returns modified n.
int modifyBit(int n, int p, int b)
{
    int mask = 1 << p;
    return (n & ~mask) | ((b << p) & mask);
}

// Driver code
int main()
{
    cout << modifyBit(6, 2, 0) << endl;
    cout << modifyBit(6, 5, 1) << endl;
    return 0;
}
```

Java

```
 // Java program to modify a bit
// at position p in n to b.
import java.io.*;

class GFG
{

// Returns modified n.
public static int modifyBit(int n,
                            int p,
                            int b)
{
    int mask = 1 << p;
    return (n & ~mask) |
           ((b << p) & mask);
}
    // Driver Code
    public static void main (String[] args)
    {
        System.out.println(modifyBit(6, 2, 0));
        System.out.println (modifyBit(6, 5, 1));
```

```
    }
}

// This code is contributed by m_kit
```

C#

```
 // C# program to modify a bit
// at position p in n to b.
using System;

class GFG
{
// Returns modified n.
public static int modifyBit(int n,
                            int p,
                            int b)
{
    int mask = 1 << p;
    return (n & ~mask) |
           ((b << p) & mask);
}

    // Driver Code
    static public void Main ()
    {

        Console.WriteLine(modifyBit(6, 2, 0));
        Console.WriteLine(modifyBit(6, 5, 1));
    }
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP program to modify a bit
// at position p in n to b.

// Returns modified n.
function modifyBit($n, $p, $b)
{
    $mask = 1 << $p;
    return ($n & ~$mask) |
           (($b << $p) & $mask);
}
```

```
    // Driver code
    echo modifyBit(6, 2, 0),"\n";
    echo modifyBit(6, 5, 1) ,"\n";

// This code is contributed by ajit
?>
```

Output :

```
 2
 38
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/modify-bit-given-position/

Chapter 225

Multiples of 4 (An Interesting Method)

Multiples of 4 (An Interesting Method) - GeeksforGeeks

Given a number n, the task is to check whether this number is a multiple of 4 or not without using +, -, * ,/ and % operators.

Examples :

```
Input: n = 4  Output - Yes
       n = 20 Output - Yes
       n = 19 Output - No
```

Method 1 (Using XOR)
An interesting fact for n > 1 is, we do XOR of all numbers from 1 to n and if the result is equal to n, then n is a multiple of 4 else not.

C++

```
 // An interesting XOR based method to check if
// a number is multiple of 4.
#include<bits/stdc++.h>
using namespace std;

// Returns true if n is a multiple of 4.
bool isMultipleOf4(int n)
{
    if (n == 1)
       return false;

    // Find XOR of all numbers from 1 to n
```

```
    int XOR = 0;
    for (int i = 1; i <= n; i++)
        XOR = XOR ^ i;

    // If XOR is equal n, then return true
    return (XOR == n);
}

// Driver code to print multiples of 4
int main()
{
    // Printing multiples of 4 using above method
    for (int n=0; n<=42; n++)
       if (isMultipleOf4(n))
         cout << n << " ";
    return 0;
}
```

Java

```
 // An interesting XOR based method to check if
// a number is multiple of 4.

class Test
{
    // Returns true if n is a multiple of 4.
    static boolean isMultipleOf4(int n)
    {
        if (n == 1)
           return false;

        // Find XOR of all numbers from 1 to n
        int XOR = 0;
        for (int i = 1; i <= n; i++)
            XOR = XOR ^ i;

        // If XOR is equal n, then return true
        return (XOR == n);
    }

    // Driver method
    public static void main(String[] args)
    {
        // Printing multiples of 4 using above method
        for (int n=0; n<=42; n++)
           System.out.print(isMultipleOf4(n) ? n : " ");
    }
}
```

Python 3

```
 # An interesting XOR based
# method to check if a
# number is multiple of 4.

# Returns true if n is a
# multiple of 4.
def isMultipleOf4(n):

    if (n == 1):
        return False

    # Find XOR of all numbers
    # from 1 to n
    XOR = 0
    for i in range(1, n + 1):
        XOR = XOR ^ i

    # If XOR is equal n, then
    # return true
    return (XOR == n)

# Driver code to print
# multiples of 4 Printing
# multiples of 4 using
# above method
for n in range(0, 43):
    if (isMultipleOf4(n)):
        print(n, end = " ")

# This code is contributed
# by Smitha
```

C#

```
 // An interesting XOR based method
// to check if a number is multiple
// of 4.
using System;
class GFG {

    // Returns true if n is a
    // multiple of 4.
    static bool isMultipleOf4(int n)
    {
        if (n == 1)
```

```
        return false;

        // Find XOR of all numbers
        // from 1 to n
        int XOR = 0;
        for (int i = 1; i <= n; i++)
            XOR = XOR ^ i;

        // If XOR is equal n, then
        // return true
        return (XOR == n);
    }

    // Driver method
    public static void Main()
    {

        // Printing multiples of 4
        // using above method
        for (int n = 0; n <= 42; n++)
        {
            if (isMultipleOf4(n))
                Console.Write(n+" ");
        }
    }
}

// This code is contributed by Smitha.
```

PHP

```
 <?php
// PHP program to check if
// a number is multiple of 4.

// Returns true if n is
// a multiple of 4.
function isMultipleOf4($n)
{
    if ($n == 1)
    return false;

    // Find XOR of all
    // numbers from 1 to n
    $XOR = 0;
    for ($i = 1; $i <= $n; $i++)
        $XOR = $XOR ^ $i;
```

```
    // If XOR is equal n,
    // then return true
    return ($XOR == $n);
}

// Driver Code

// Printing multiples of 4
// using above method
for ($n = 0; $n <= 42; $n++)
if (isMultipleOf4($n))
    echo $n, " ";

// This code is contribued by Ajit
?>
```

Output :

```
0 4 8 12 16 20 24 28 32 36 40
```

How does this work?
When we do XOR of numbers, we get 0 as XOR value just before a multiple of 4. This keeps repeating before every multiple of 4.

```
Number Binary-Repr  XOR-from-1-to-n
1         1           [0001]
2        10           [0011]
3        11           [0000]
```

Method 2 (Using Bitwise Shift Operators)
The idea is to remove last two bits using >>, then multiply with 4 using <<. If final result is same as n, then last two bits were 0, hence number was a multiple of four.

C++

```
 // An interesting XOR based method to check if
// a number is multiple of 4.
#include<bits/stdc++.h>
using namespace std;

// Returns true if n is a multiple of 4.
bool isMultipleOf4(long long n)
{
    if (n==0)
```

```
        return true;

    return (((n>>2)<<2) == n);
}

// Driver code to print multiples of 4
int main()
{
    // Printing multiples of 4 using above method
    for (int n=0; n<=42; n++)
        if (isMultipleOf4(n))
            cout << n << " ";
    return 0;
}
```

Java

```
 // An interesting XOR based method to check if
// a number is multiple of 4.

class Test
{
    // Returns true if n is a multiple of 4.
    static boolean isMultipleOf4(long n)
    {
        if (n==0)
            return true;

        return (((n>>2)<<2) == n);
    }

    // Driver method
    public static void main(String[] args)
    {
        // Printing multiples of 4 using above method
        for (int n=0; n<=42; n++)
           System.out.print(isMultipleOf4(n) ? n : " ");
    }
}
```

C#

```
 // An interesting XOR based method to
// check if a number is multiple of 4.
using System;

class GFG {
```

```
    // Returns true if n is a multiple
    // of 4.
    static bool isMultipleOf4(int n)
    {
        if (n == 0)
            return true;

        return (((n >> 2) << 2) == n);
    }

    // Driver code to print multiples
    // of 4
    static void Main()
    {

        // Printing multiples of 4 using
        // above method
        for (int n = 0; n <= 42; n++)
            if (isMultipleOf4(n))
                Console.Write(n + " ");
    }
}

// This code is contributed by Anuj_67
```

PHP

```
 <?php
// PHP program to check if
// a number is multiple of 4.

// Returns true if n is
// a multiple of 4.
function isMultipleOf4($n)
{
    if ($n == 0)
        return true;

    return ((($n >> 2) << 2) == $n);
}

// Driver Code

// Printing multiples of 4
// using above method
for ($n = 0; $n <= 42; $n++)
    if (isMultipleOf4($n))
```

```
        echo $n , " ";

// This code is contributed by anuj_67.
?>
```

Output :

```
0 4 8 12 16 20 24 28 32 36 40
```

As we can see that the main idea to find multiplicity of 4 is to check the least two significant bits of the given number. We know that for any even number, the least significant bit is always ZERO (i.e. 0). Similarly, for any number which is multiple of 4 will have least two significant bits as ZERO. And with the same logic, for any number to be multiple of 8, least three significant bits will be ZERO. That's why we can use AND operator (&) as well with other operand as 0x3 to find multiplicity of 4.

Improved By : vt_m, jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/multiples-4-interesting-method/

Chapter 226

Multiplication of two numbers with shift operator

Multiplication of two numbers with shift operator - GeeksforGeeks

For any given two numbers n and m, you have to find n*m without using any multiplication operator.

Examples :

```
Input: n = 25 , m = 13
Output: 325

Input: n = 50 , m = 16
Output: 800
```

We can solve this problem with the shift operator. The idea is based on the fact that every number can be represented in binary form. And multiplication with a number is equivalent to multiplication with powers of 2. Powers of 2 can be obtained using left shift operator.

Check for every set bit in the binary representation of m and for every set bit left shift n, count times where count if place value of the set bit of m and add that value to answer.

C++

```
 // CPP program to find multiplication
// of two number without use of
// multiplication operator
#include<bits/stdc++.h>
using namespace std;

// Function for multiplication
int multiply(int n, int m)
```

```
{
    int ans = 0, count = 0;
    while (m)
    {
        // check for set bit and left
        // shift n, count times
        if (m % 2 == 1)
            ans += n << count;

        // increment of place value (count)
        count++;
        m /= 2;
    }
    return ans;
}

// Driver code
int main()
{
    int n = 20 , m = 13;
    cout << multiply(n, m);
    return 0;
}
```

Java

```
 // Java program to find multiplication
// of two number without use of
// multiplication operator
class GFG
{

    // Function for multiplication
    static int multiply(int n, int m)
    {
        int ans = 0, count = 0;
        while (m > 0)
        {
            // check for set bit and left
            // shift n, count times
            if (m % 2 == 1)
                ans += n << count;

            // increment of place
            // value (count)
            count++;
            m /= 2;
        }
```

```
        return ans;
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 20, m = 13;

        System.out.print( multiply(n, m) );
    }
}

// This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to find multiplication
// of two number without use of
// multiplication operator
using System;

class GFG
{

    // Function for multiplication
    static int multiply(int n, int m)
    {
        int ans = 0, count = 0;
        while (m > 0)
        {
            // check for set bit and left
            // shift n, count times
            if (m % 2 == 1)
                ans += n << count;

            // increment of place
            // value (count)
            count++;
            m /= 2;
        }

        return ans;
    }

    // Driver Code
    public static void Main ()
    {
```

```
        int n = 20, m = 13;

        Console.WriteLine( multiply(n, m) );
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find multiplication
// of two number without use of
// multiplication operator

// Function for multiplication
function multiply( $n, $m)
{
    $ans = 0; $count = 0;
    while ($m)
    {
        // check for set bit and left
        // shift n, count times
        if ($m % 2 == 1)
            $ans += $n << $count;

        // increment of place value (count)
        $count++;
        $m /= 2;
    }
    return $ans;
}

// Driver code
$n = 20 ; $m = 13;
echo multiply($n, $m);

// This code is contributed by anuj_67.
?>
```

Output :

```
260
```

Time Complexity : O(log n)

Related Article:
Russian Peasant (Multiply two numbers using bitwise operators)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/multiplication-two-numbers-shift-operator/

Chapter 227

Multiplication with a power of 2

Multiplication with a power of 2 - GeeksforGeeks

Given two numbers x and n, we need to multiply x with 2^n

Examples :

```
Input  : x = 25, n = 3
Output : 200
25 multiplied by 2 raised to power 3
is 200.

Input : x = 70, n = 2
Output : 280
```

A **simple solution** is to compute n-th power of 2 and then multiply with x.

C++

```
 // Simple C/C++ program
// to compute x * (2^n)
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;

// Returns 2 raised to power n
ll power2(ll n)
{
    if (n == 0)
        return 1;
```

```
    if (n == 1)
        return 2;

    return power2(n / 2) *
                    power2(n / 2);
}

ll multiply(ll x, ll n)
{
    return x * power2(n);
}

// Driven program
int main()
{
    ll x = 70, n = 2;
    cout<<multiply(x, n);
    return 0;
}
```

Java

```
 // Simple Java program
// to compute x * (2^n)
import java.util.*;

class GFG {

    // Returns 2 raised to power n
    static long power2(long n)
    {
        if (n == 0)
            return 1;

        if (n == 1)
            return 2;

        return power2(n / 2)
                            * power2(n / 2);
    }

    static long multiply(long x, long n)
    {
        return x * power2(n);
    }

    /* Driver program */
    public static void main(String[] args)
```

```
    {
        long x = 70, n = 2;

        System.out.println(multiply(x, n));
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Simple Python program
# to compute x * (2^n)

# Returns 2 raised to power n
def power2(n):

    if (n == 0):
        return 1
    if (n == 1):
        return 2
    return power2(n / 2) *
                  power2(n / 2);


def multiply(x, n):
    return x * power2(n);


# Driven program
x = 70
n = 2
print(multiply(x, n))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // Simple C# program
// to compute x * (2^n)
using System;

class GFG {

    // Returns 2 raised to power n
    static long power2(long n)
    {
```

```
        if (n == 0)
            return 1;

        if (n == 1)
            return 2;

        return power2(n / 2)
                        * power2(n / 2);
    }

    static long multiply(long x, long n)
    {
        return x * power2(n);
    }

    /* Driver program */
    public static void Main()
    {
        long x = 70, n = 2;

        Console.WriteLine(multiply(x, n));
    }
}

// This code is contributed by Vt_m.
```

PHP

```
 <?php
// Simple PHP program
// to compute x * (2^n)

// Returns 2 raised to power n
function power2($n)
{
    if ($n == 0)
        return 1;

    if ($n == 1)
        return 2;

    return power2($n / 2) *
           power2($n / 2);
}

function multiply( $x, $n)
{
    return $x * power2($n);
```

```
}

// Driver Code
$x = 70; $n = 2;
echo multiply($x, $n);

// This code is contributed by ajit
?>
```

Output :

```
280
```

Time complexity : O(Log n)

An **efficient solution** is to usebitwise leftshift operator. We know 1 << n means 2 raised to power n.

C++

```
 // Efficient C/C++ program to compute x * (2^n)
#include <stdio.h>
typedef long long int ll;

ll multiply(ll x, ll n)
{
    return x << n;
}

// Driven program to check above function
int main()
{
    ll x = 70, n = 2;
    printf("%lld", multiply(x, n));
    return 0;
}
```

Java

```
 // JAVA Code for Multiplication with a
// power of 2
import java.util.*;

class GFG {

    static long multiply(long x, long n)
    {
```

```
        return x << n;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        long x = 70, n = 2;
        System.out.println(multiply(x, n));
    }
}

//This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Efficient Python3 code to compute x * (2^n)

def multiply( x , n ):
    return x << n

# Driven code to check above function
x = 70
n = 2
print( multiply(x, n))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# Code for Multiplication with a
// power of 2
using System;

class GFG {

    static int multiply(int x, int n)
    {
        return x << n;
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int x = 70, n = 2;

        Console.WriteLine(multiply(x, n));
    }
```

```
}

//This code is contributed by vt_m.
```

PHP

```
 <?php
// Efficient PHP program to compute x * (2^n)

function multiply($x, $n)
{
    return $x << $n;
}

    // Driver Code
    $x = 70;
    $n = 2;
    echo multiply($x, $n);

// This code is contributed by ajit
?>
```

Output :

```
280
```

Time complexity : O(1)

Improved By : jit_t

Source

https://www.geeksforgeeks.org/multiplication-power-2/

Chapter 228

Multiply a given Integer with 3.5

Multiply a given Integer with 3.5 - GeeksforGeeks

Given a integer x, write a function that multiplies x with 3.5 and returns the integer result. You are not allowed to use %, /, *.

```
Examples :
Input: 2
Output: 7

Input: 5
Output: 17 (Ignore the digits after decimal point)
```

Solution:
1. We can get x*3.5 by adding 2*x, x and x/2. To calculate 2*x, left shift x by 1 and to calculate x/2, right shift x by 2.

C++

```
 // C++ program to multiply
// a number with 3.5
#include <bits/stdc++.h>

int multiplyWith3Point5(int x)
{
    return (x<<1) + x + (x>>1);
}

/* Driver program to test above functions*/
```

```
int main()
{
    int x = 4;
    printf("%d", multiplyWith3Point5(x));
    getchar();
    return 0;
}
```

Java

```
 // Java Program to multiply
// a number with 3.5

class GFG {

    static int multiplyWith3Point5(int x)
    {
        return (x<<1) + x + (x>>1);
    }

    /* Driver program to test above functions*/
    public static void main(String[] args)
    {
        int x = 2;
        System.out.println(multiplyWith3Point5(x));
    }
}

// This code is contributed by prerna saini.
```

Python3

```
 # Python 3 program to multiply
# a number with 3.5

def multiplyWith3Point5(x):

    return (x<<1) + x + (x>>1)


# Driver program to
# test above functions
x = 4
print(multiplyWith3Point5(x))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# Program to multiply
// a number with 3.5
using System;

class GFG
{

    static int multiplyWith3Point5(int x)
    {
        return (x<<1) + x + (x>>1);
    }

    /* Driver program to test above functions*/
    public static void Main()
    {
        int x = 2;
        Console.Write(multiplyWith3Point5(x));
    }

}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to multiply
// a number with 3.5


function multiplyWith3Point5( $x)
{
    return ($x << 1) + $x + ($x >> 1);
}

// Driver Code
$x = 4;
echo multiplyWith3Point5($x);

// This code is contributed by vt_m.
?>
```

2. Another way of doing this could be (8*x – x)/2 (See below code). Thanks to *Ajaym* for suggesting this.

```
 #include <stdio.h>
int multiplyWith3Point5(int x)
{
  return ((x<<3) - x)>>1;
}
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/multiply-an-integer-with-3-5/

Chapter 229

Multiply a number with 10 without using multiplication operator

Multiply a number with 10 without using multiplication operator - GeeksforGeeks

Given a number, the task is to multiply it with 10 without using multiplication operator?

Examples:

```
Input : n = 50
Output: 500
// multiplication of 50 with 10 is = 500

Input : n = 16
Output: 160
// multiplication of 16 with 10 is = 160
```

A **simple solution** for this problem is to run a loop and add n with itself 10 times. Here we need to perform 10 operations.
A **better solution** is to use bit manipulation. We have to multiply n with 10 i.e; n*10, we can write this as **n*(2+8) = n*2 + n*8** and since we are not allowed to use multiplication operator we can do this using left shift bitwise operator. So n*10 = n<<1 + n<<3.

C++

```
 // C++ program to multiply a number with 10 using
// bitwise operators
#include<bits/stdc++.h>
using namespace std;
```

```
// Function to find multiplication of n with
// 10 without usng multiplication operator
int multiplyTen(int n)
{
    return (n<<1) + (n<<3);
}

// Driver program to run the case
int main()
{
    int n = 50;
    cout << multiplyTen(n);
    return 0;
}
```

Java

```
 // Java Code to Multiply a number with 10
// without using multiplication operator
import java.util.*;

class GFG {

    // Function to find multiplication of n
    // with 10 without usng multiplication
    // operator
    public static int multiplyTen(int n)
    {
        return (n << 1) + (n << 3);
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int n = 50;
        System.out.println(multiplyTen(n));

    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python 3

```
 # Python 3 program to multiply a
# number with 10 using bitwise
# operators
```

```
# Function to find multiplication
# of n with 10 without usng
# multiplication operator
def multiplyTen(n):

    return (n << 1) + (n << 3)

# Driver program to run the case
n = 50
print (multiplyTen(n))

# This code is contributed by
# Smitha
```

C#

```
 // C# Code to Multiply a number with 10
// without using multiplication operator
using System;

class GFG {

    // Function to find multiplication of n
    // with 10 without usng multiplication
    // operator
    public static int multiplyTen(int n)
    {
        return (n << 1) + (n << 3);
    }

    // Driver Code
    public static void Main()
    {
        int n = 50;
        Console.Write(multiplyTen(n));

    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP program to multiply a
// number with 10 using
```

```
// bitwise operators

// Function to find multiplication
// of n with 10 without using
// multiplication operator
function multiplyTen($n)
{
    return ($n << 1) + ($n << 3);
}

    // Driver Code
    $n = 50;
    echo multiplyTen($n);

// This code is contributed by nitin mittal.
?>
```

Output:

```
500
```

Improved By : nitin mittal, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/multiply-number-10-without-using-multiplication-operator/

Chapter 230

Multiply any Number with 4 using Bitwise Operator

Multiply any Number with 4 using Bitwise Operator - GeeksforGeeks

We are a Number n and our task is multiply the number with 4 using bit-wise Operator.

Examples:

```
Input : 4
Output :16

Input :5
Output :20
```

Explanation Case 1:- n=4 the binary of 4 is 100 and now shift two bit right then 10000 now the number is 16 that is multiply 4*4=16 ans.
Approach :- (n<<2) shift two bit right

C++

```
 // C++ program to multiply a number with
// 4 using Bitwise Operator
#include <bits/stdc++.h>
using namespace std;

// function the return multiply a number
//  with 4 using bitwise operator
int multiplyWith4(int n)
{
    // returning a number with multipy
    // with 4 using2 bit shifring right
```

```
    return (n << 2);
}

// derive function
int main()
{
    int n = 4;
    cout << multiplyWith4(n) << endl;
    return 0;
}
```

Java

```
 // Java program to multiply a number
// with 4 using Bitwise Operator

class GFG {

// function the return
// multiply a number
// with 4 using bitwise
// operator
static int multiplyWith4(int n)
{

    // returning a number
    // with multipy with
    // 4 using 2 bit shifting
    // right
    return (n << 2);
}

// Driver Code
public static void main(String[] args)
{
    int n = 4;
    System.out.print(multiplyWith4(n));
}
}

// This code is contributed by Smitha.
```

Python 3

```
 # Python 3 program to multiply
# a number with 4 using Bitwise
# Operator
```

```
# function the return multiply
# a number with 4 using bitwise
# operator
def multiplyWith4(n):

    # returning a number with
    # multipy with 4 using2
    # bit shifring right
    return (n << 2)

# derive function
n = 4
print(multiplyWith4(n))

# This code is contributed
# by Smitha
```

C#

```
 // C# program to multiply a number
// with 4 using Bitwise Operator
using System;

class GFG {

// function the return
// multiply a number
// with 4 using bitwise
// operator
static int multiplyWith4(int n)
{

    // returning a number
    // with multipy with
    // 4 using 2 bit shifting
    // right
    return (n << 2);
}

// Driver Code
public static void Main(String[] args)
{
    int n = 4;
    Console.Write(multiplyWith4(n));
}
}
```

```
// This code is contributed by Smitha.
```

PHP

```
 <?php
// PHP program to multiply
// a number with 4 using
// Bitwise Operator

// function the return
// multiply a number
// with 4 using bitwise
// operator
function multiplyWith4($n)
{

    // returning a number
    // with multipy with
    // 4 using2 bit
    // shifting right
    return ($n << 2);
}

    // Driver Code
    $n = 4;
    echo multiplyWith4($n),"\n";

// This code is contributed by Ajit.
?>
```

Output :-

```
16
```

Generalization : In general, we can multiply with a power of 2 using bitwise operators. For example, suppose we wish to multiply with 16 (which is 2^4), we can do it by left shifting by 4.

Improved By : Smitha Dinesh Semwal, jit_t

Source

https://www.geeksforgeeks.org/multiply-number-4-using-bitwise-operator/

Chapter 231

Multiplying a variable with a constant without using multiplication operator

Multiplying a variable with a constant without using multiplication operator - GeeksforGeeks

As we know that every number can be represented as sum(or difference) of powers of 2, therefore what we can do is represent the constant as a sum of powers of 2.

For this purpose we can use the bitwise left shift operator. When a number is bitwise left shifted it is multiplied by 2 for every bit shift.

For example, suppose we want to multiply a variable say “a” by 10 then what we can do is

```
a = a << 3 + a << 1;
```

The expression a << 3 multiplies a by 8 ans expression a<<1 multiplies it by 2.

So basically what we have here is a = a*8 + a*2 = a*10

Similarly for multiplying with 7 what we can do is

```
a = a<<3 - a;
or
a = a<<2 + a<<1 + a;
```

Both these statements multiply a by 7.

C++

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

// Returns n * 7
int multiplyBySeven(int n)
{
    // OR (n << 2) + (n << 1) + n
    return (n << 3) - n;
}

// Returns n * 12
int multiplyByTwelve(int n)
{
    return (n << 3) + (n << 2);
}

int main()
{
    cout << multiplyBySeven(5) << endl;
    cout << multiplyByTwelve(5) << endl;
    return 0;
}
```

Java

```
 class GFG {

    // Returns n * 7
    static int multiplyBySeven(int n)
    {

        // OR (n << 2) + (n << 1) + n
        return (n << 3) - n;
    }

    // Returns n * 12
    static int multiplyByTwelve(int n)
    {
        return (n << 3) + (n << 2);
    }

    // Driver code
    public static void main(String[] args)
    {
        System.out.println(multiplyBySeven(5));
        System.out.println(multiplyByTwelve(5));
    }
}
```

```
// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to Multiplying a
# variable with a constant

# Returns n * 7
def multiplyBySeven(n):

    # OR (n << 2) + (n << 1) + n
    return (n << 3) - n

# Returns n * 12
def multiplyByTwelve(n):
    return (n << 3) + (n << 2)

# Driver code
print(multiplyBySeven(5))
print(multiplyByTwelve(5))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to Multiplying a
// variable with a constant
using System;

class GFG
{
    // Returns n * 7
    static int multiplyBySeven(int n)
    {
        // OR (n << 2) + (n << 1) + n
        return (n << 3) - n;
    }

    // Returns n * 12
    static int multiplyByTwelve(int n)
    {
        return (n << 3) + (n << 2);
    }

    // Driver code
    public static void Main()
```

```
    {
        Console.WriteLine(multiplyBySeven(5));
        Console.WriteLine(multiplyByTwelve(5));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program of multiply operator
// Returns n * 7

function multiplyBySeven($n)
{
    return ($n << 3) - $n;
}

// Returns n * 12
function multiplyByTwelve($n)
{
    return ($n << 3) + ($n << 2);
}

// Driver Code
echo multiplyBySeven(5), "\n";
echo multiplyByTwelve(5), "\n";

// This code is contributed by Ajit
?>
```

Output :

```
35
60
```

We just need to find the combination of powers of 2. Also, this comes really handy when we have a very large dataset and each one of them requires multiplication with the same constant as bitwise operators are faster as compared to mathematical operators.

Improved By : jit_t

Source

https://www.geeksforgeeks.org/multiplying-variable-constant-without-using-multiplication-operator/

Chapter 232

Next greater integer having one more number of set bits

Next greater integer having one more number of set bits - GeeksforGeeks

Given a positive integer 'n' having 'x' number of set bits in its binary representation. The problem is to find the next greater integer(smallest integer greater than n), having (x+1) number of set bits in its binary representation.

Examples :

```
Input : 10
Output : 11
(10)10 = (1010)2
is having 2 set bits.

(11)10 = (1011)2
is having 3 set bits and is the next greater.

Input : 39
Output : 47
```

Approach: Following are the steps:

1. Find the position of the rightmost unset bit(considering last bit at position 0, second last bit at position 1 and so on) in the binary representation of **n**.
2. Let the position be represented by **pos**.
3. Set the bit at position **pos**. Refer this post.
4. If there are no unset bits in the binary representation, then perform bitwise left shift by 1 on the given number and then add 1 to it.

How to get the position of rightmost unset bit?

1. Perform bitwise not on the given number(operation equivalent to 1's complement).Let it be **num** = ~n.
2. Get the position of rightmost set bit of **num**.

C++

```
 // C++ implementation to find the next greater integer
// with one more number of set bits
#include <bits/stdc++.h>

using namespace std;

// function to find the position of rightmost
// set bit. Returns -1 if there are no set bits
int getFirstSetBitPos(int n)
{
    return (log2(n&-n)+1) - 1;
}

// function to find the next greater integer
int nextGreaterWithOneMoreSetBit(int n)
{
    // position of rightmost unset bit of n
    // by passing ~n as argument
    int pos = getFirstSetBitPos(~n);

    // if n consists of unset bits, then
    // set the rightmost unset bit
    if (pos > -1)
        return (1 << pos) | n;

    //n does not consists of unset bits
    return ((n << 1) + 1);
}

// Driver program to test above
int main()
{
    int n = 10;
    cout << "Next greater integer = "
          << nextGreaterWithOneMoreSetBit(n);
    return 0;
}
```

Java

```
 // Java implementation to find the next greater integer
// with one more number of set bits
class GFG {

    // function to find the position of rightmost
    // set bit. Returns -1 if there are no set bits
    static int getFirstSetBitPos(int n)
    {
        return ((int)(Math.log(n & -n) / Math.log(2)) + 1) - 1;
    }

    // function to find the next greater integer
    static int nextGreaterWithOneMoreSetBit(int n)
    {

        // position of rightmost unset bit of n
        // by passing ~n as argument
        int pos = getFirstSetBitPos(~n);

        // if n consists of unset bits, then
        // set the rightmost unset bit
        if (pos > -1)
            return (1 << pos) | n;

        // n does not consists of unset bits
        return ((n << 1) + 1);
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 10;
        System.out.print("Next greater integer = "
                + nextGreaterWithOneMoreSetBit(n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 implementation to find
# the next greater integer with
# one more number of set bits
import math

# Function to find the position
# of rightmost set bit. Returns -1
```

```
# if there are no set bits
def getFirstSetBitPos(n):

    return ((int)(math.log(n & -n) /
                  math.log(2)) + 1) - 1

# Function to find the next greater integer
def nextGreaterWithOneMoreSetBit(n):

    # position of rightmost unset bit of
    # n by passing ~n as argument
    pos = getFirstSetBitPos(~n)

    # if n consists of unset bits, then
    # set the rightmost unset bit
    if (pos > -1):
        return (1 << pos) | n

    # n does not consists of unset bits
    return ((n << 1) + 1)

# Driver code
n = 10
print("Next greater integer = ",
       nextGreaterWithOneMoreSetBit(n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# implementation to find the next greater
// integer with one more number of set bits
using System;

class GFG {

    // function to find the position of rightmost
    // set bit. Returns -1 if there are no set bits
    static int getFirstSetBitPos(int n)
    {
        return ((int)(Math.Log(n & -n) / Math.Log(2))
                                            + 1) - 1;
    }

    // function to find the next greater integer
    static int nextGreaterWithOneMoreSetBit(int n)
    {
```

```
        // position of rightmost unset bit of n
        // by passing ~n as argument
        int pos = getFirstSetBitPos(~n);

        // if n consists of unset bits, then
        // set the rightmost unset bit
        if (pos > -1)
            return (1 << pos) | n;

        // n does not consists of unset bits
        return ((n << 1) + 1);
    }

    // Driver code
    public static void Main()
    {
        int n = 10;

        Console.Write("Next greater integer = "
              + nextGreaterWithOneMoreSetBit(n));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP implementation to find the
// next greater integer with
// one more number of set bits

// function to find the position
// of rightmost set bit. Returns
// -1 if there are no set bits

function getFirstSetBitPos($n)
{
    return (log($n & -$n + 1)) - 1;
}

// function to find the
// next greater integer

function nextGreaterWithOneMoreSetBit($n)
{
    // position of rightmost unset bit of n
    // by passing ~n as argument
```

```
    $pos = getFirstSetBitPos(~$n);

    // if n consists of unset bits, then
    // set the rightmost unset bit
    if ($pos > -1)
        return (1 << $pos) | $n;

    //n does not consists of unset bits
    return (($n << 1) + 1);
}

// Driver Code
$n = 10;
echo "Next greater integer = ",
    nextGreaterWithOneMoreSetBit($n);

// This code is contributed by Ajit
?>
```

Output :

```
Next greater integer = 11
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/next-greater-integer-one-number-set-bits/

Chapter 233

Next higher number with same number of set bits

Next higher number with same number of set bits - GeeksforGeeks

Given a number x, find next number with same number of 1 bits in it's binary representation.

For example, consider x = 12, whose binary representation is 1100 (excluding leading zeros on 32 bit machine). It contains two logic 1 bits. The next higher number with two logic 1 bits is 17 (10001_2).

Algorithm:

When we observe the binary sequence from 0 to $2^n - 1$ (n is # of bits), right most bits (least significant) vary rapidly than left most bits. The idea is to find right most string of 1's in x, and shift the pattern to right extreme, except the left most bit in the pattern. Shift the left most bit in the pattern (omitted bit) to left part of x by one position. An example makes it more clear,

```
x = 156
```

10

```
x = 10011100
```

(2)

```
10011100
00011100 - right most string of 1's in x
00000011 - right shifted pattern except left most bit ------> [A]
00010000 - isolated left most bit of right most 1's pattern
00100000 - shiftleft-ed the isolated bit by one position ------> [B]
10000000 - left part of x, excluding right most 1's pattern ------> [C]
10100000 - add B and C (OR operation) ------> [D]
10100011 - add A and D which is required number 163
```

(10)

After practicing with few examples, it easy to understand. Use the below given program for generating more sets.

Program Design:

We need to note few facts of binary numbers. The expression x & -x will isolate right most set bit in x (ensuring x will use 2's complement form for negative numbers). If we add the result to x, right most string of 1's in x will be reset, and the immediate '0' left to this pattern of 1's will be set, which is part [B] of above explanation. For example if x = 156, x & -x will result in 00000100, adding this result to x yields 10100000 (see part D). We left with the right shifting part of pattern of 1's (part A of above explanation).

There are different ways to achieve part A. Right shifting is essentially a division operation. What should be our divisor? Clearly, it should be multiple of 2 (avoids 0.5 error in right shifting), and it should shift the right most 1's pattern to right extreme. The expression (x & -x) will serve the purpose of divisor. An EX-OR operation between the number X and expression which is used to reset right most bits, will isolate the rightmost 1's pattern.

A Correction Factor:

Note that we are adding right most set bit to the bit pattern. The addition operation causes a shift in the bit positions. The weight of binary system is 2, one shift causes an increase by a factor of 2. Since the increased number (*rightOnesPattern* in the code) being used twice, the error propagates twice. The error needs to be corrected. A right shift by 2 positions will correct the result.

The popular name for this program is **s**ame **n**umber **o**f **o**ne **b**its.

C++

```
 #include<iostream>

using namespace std;

typedef unsigned int uint_t;

// this function returns next higher number with same number of set bits as x.
uint_t snoob(uint_t x)
{

  uint_t rightOne;
  uint_t nextHigherOneBit;
  uint_t rightOnesPattern;

  uint_t next = 0;

  if(x)
  {
```

```
    // right most set bit
    rightOne = x & -(signed)x;

    // reset the pattern and set next higher bit
    // left part of x will be here
    nextHigherOneBit = x + rightOne;

    // nextHigherOneBit is now part [D] of the above explanation.

    // isolate the pattern
    rightOnesPattern = x ^ nextHigherOneBit;

    // right adjust pattern
    rightOnesPattern = (rightOnesPattern)/rightOne;

    // correction factor
    rightOnesPattern >>= 2;

    // rightOnesPattern is now part [A] of the above explanation.

    // integrate new pattern (Add [D] and [A])
    next = nextHigherOneBit | rightOnesPattern;
  }

  return next;
}

int main()
{
  int x = 156;
  cout<<"Next higher number with same number of set bits is "<<snoob(x);

  getchar();
  return 0;
}
```

Python 3

```
 # This function returns next
# higher number with same
# number of set bits as x.
def snoob(x):

    next = 0
    if(x):

        # right most set bit
        rightOne = x & -(x)
```

```
        # reset the pattern and
        # set next higher bit
        # left part of x will
        # be here
        nextHigherOneBit = x + int(rightOne)

        # nextHigherOneBit is
        # now part [D] of the
        # above explanation.
        # isolate the pattern
        rightOnesPattern = x ^ int(nextHigherOneBit)

        # right adjust pattern
        rightOnesPattern = (int(rightOnesPattern) /
                            int(rightOne))

        # correction factor
        rightOnesPattern = int(rightOnesPattern) >> 2

        # rightOnesPattern is now part
        # [A] of the above explanation.

        # integrate new pattern
        # (Add [D] and [A])
        next = nextHigherOneBit | rightOnesPattern
    return next

# Driver Code
x = 156
print("Next higher number with " +
      "same number of set bits is",
                        snoob(x))

# This code is contributed by Smita
```

Usage: Finding/Generating subsets.

Variations:

1. Write a program to find a number immediately smaller than given, with same number of logic 1 bits? (Pretty simple)
2. How to count or generate the subsets available in the given set?

References:

1. A nice presentation here.

2. Hackers Delight by Warren (An excellent and short book on various bit magic algorithms, a must for enthusiasts)
3. C A Reference Manual by Harbison and Steele (A good book on standard C, you can access code part of this post here).

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

Improved By : Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/next-higher-number-with-same-number-of-set-bits/

Chapter 234

Number of Reflexive Relations on a Set

Number of Reflexive Relations on a Set - GeeksforGeeks

Given a number n, find out number of Reflexive Relation on a set of first n natural numbers {1, 2, ..n}.

Examples :

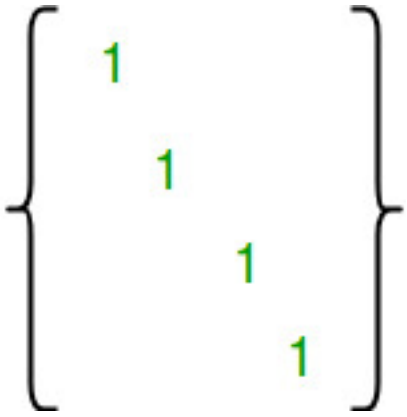
```
Input  : n = 2
Output : 4
The given set A = {1, 2}. The following
are reflexive relations on A * A :
{{1, 1), (2, 2)}
{(1, 1), (2, 2), (1, 2)}
{(1, 1), (2, 2), (1, 2), (2, 1)}
{(1, 1), (2, 2), (2, 1)}

Input  : n = 3
Output : 64
The given set is {1, 2, 3}. There are
64 reflexive relations on A * A :
```

Explanation :

Reflexive Relation : A Relation R on A a set A is said to be Reflexive if xRx for every element of x ? A.

The number of reflexive relations on an n-element set is $\mathbf{2^{n^2 - n}}$

How does this formula work?
A relation R is reflexive if the matrix diagonal elements are 1.



If we take a closer look the matrix, we can notice that the size of matrix is n^2. The n diagonal entries are fixed. For remaining $n^2 - n$ entries, we have choice to either fill 0 or 1. So there are total $\mathbf{2^{n^2 - n}}$ ways of filling the matrix.

CPP

```
 // C++ Program to count reflexive relations
// on a set of first n natural numbers.
#include <iostream>
using namespace std;

int countReflexive(int n)
{
   // Return 2^(n*n - n)
   return (1 << (n*n - n));
}

int main()
{
    int n = 3;
    cout << countReflexive(n);
    return 0;
}
```

Java

```
 // Java Program to count reflexive
// relations on a set of first n
// natural numbers.

import java.io.*;
import java.util.*;

class GFG {
```

```
static int countReflexive(int n)
{

// Return 2^(n*n - n)
return (1 << (n*n - n));

}

// Driver function
    public static void main (String[] args) {
    int n = 3;
    System.out.println(countReflexive(n));

    }
}

// This code is contributed by Gitanjali.
```

Python3

```
 # Python3 Program to count
# reflexive relations
# on a set of first n
# natural numbers.


def countReflexive(n):

    # Return 2^(n*n - n)
    return (1 << (n*n - n));

# driver function
n = 3
ans = countReflexive(n);
print (ans)

# This code is contributed by saloni1297
```

C#

```
 // C# Program to count reflexive
// relations on a set of first n
// natural numbers.
using System;

class GFG {
```

```
    static int countReflexive(int n)
    {

        // Return 2^(n*n - n)
        return (1 << (n*n - n));

    }

    // Driver function
    public static void Main () {

    int n = 3;
    Console.WriteLine(countReflexive(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to count
// reflexive relations on a
// set of first n natural numbers.

function countReflexive($n)
{
// Return 2^(n * n - n)
return (1 << ($n * $n - $n));
}

//Driver code
$n = 3;
echo countReflexive($n);

// This code is contributed by mits
?>
```

Output :

```
64
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/number-reflexive-relations-set/

Chapter 235

Number of integers with odd number of set bits

Number of integers with odd number of set bits - GeeksforGeeks

Given a number n, count number of integers smaller than or equal to n that have odd number of set bits.

Examples:

```
Input : 5
Output : 3
Explanation :
Integers with odd number of
set bits in range 1 to 5 :
0 contains 0 set bits
1 contains 1 set bits
2 contains 1 set bits
3 contains 2 set bits
4 contains 1 set bits
5 contains 2 set bits

Input : 10
Output : 5
Explanation :
Integers with odd set bits are 1, 2,
4, 7 and 8.
```

Prerequisites : Count number of set bits

The idea is based on below fact.

> If n is odd then there are total n+1 integers smaller than or equal to n (0, 1, 2 … n) and half of these integers contain odd number of set bits.

How to handle case when n is even? We know result for n-1. We count set bits in n and add 1 to n/2 if the count is odd. Else we return n/2.

C++

```
 // CPP code to find numbers with
// odd number of set bits
#include <bits/stdc++.h>
using namespace std;

// function that returns the number
// of integers with odd number of
// set bits
int countWithOddSetBits(int n)
{
    // If n is odd, then half of the
    // integers in (0, 1, .. n) contain
    // odd number of set bits.
    if (n % 2 != 0)
        return (n + 1) / 2;

    // If n is even, we know result for
    // n-1. We explicitly compute set bit
    // count in n.
    int count = __builtin_popcount(n);

    int ans = n / 2;
    if (count % 2 != 0)
        ans++;
    return ans;
}

// Driver code
int main()
{
    int n = 10;
    cout << countWithOddSetBits(n);
    return 0;
}
```

Java

```
 // Java code to find numbers
// with odd number of set bits
import java.io.*;
```

```
class GFG
{

// function that returns the
// number of integers with
// odd number of set bits
static int countWithOddSetBits(int n)
{
    // If n is odd, then half
    // of the integers in
    // (0, 1, .. n) contain
    // odd number of set bits.
    if (n % 2 != 0)
        return (n + 1) / 2;

    // If n is even, we know
    // result for n-1. We
    // explicitly compute set
    // bit count in n.
    int count = (n);

    int ans = n / 2;
    if (count % 2 != 0)
        ans++;
    return ans;
}

// Driver Code
public static void main (String[] args)
{
    int n = 10;
    System.out.println( countWithOddSetBits(n));
}
}

// This code is contributed by aj_36
```

C#

```
 // C# code to find numbers
// with odd number of set bits
using System;

class GFG
{

// function that returns the
```

```
// number of integers with
// odd number of set bits
static int countWithOddSetBits(int n)
{
    // If n is odd, then half
    // of the integers in
    // (0, 1, .. n) contain
    // odd number of set bits.
    if (n % 2 != 0)
        return (n + 1) / 2;

    // If n is even, we know
    // result for n-1. We
    // explicitly compute set
    // bit count in n.
    int count = (n);

    int ans = n / 2;
    if (count % 2 != 0)
        ans++;
    return ans;
}

// Driver Code
static public void Main ()
{
    int n = 10;
    Console.WriteLine(countWithOddSetBits(n));
}
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP code to find numbers with
// odd number of set bits

// function that returns the number
// of integers with odd number of
// set bits
function countWithOddSetBits($n)
{
    // If n is odd, then half of
    // the integers in (0, 1, .. n)
    // contain odd number of set bits.
    if ($n % 2 != 0)
```

```
        return ($n + 1) / 2;

    // If n is even, we know result
    // for n-1. We explicitly compute
    // set bit count in n.
    $count = ($n);

    $ans = $n / 2;
    if ($count % 2 != 0)
        $ans++;
    return $ans;
}

// Driver code
$n = 10;
echo countWithOddSetBits($n);

// This code is contributed by aj_36
?>
```

Output :

```
5
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/number-integers-odd-number-set-bits/

Chapter 236

Number of pairs with Pandigital Concatenation

Number of pairs with Pandigital Concatenation - GeeksforGeeks

A pair of strings when concatenated is said to be a 'Pandigital Concatenation' if their concatenation consists of all digits from (0 – 9) in any order at least once.The task is, given N strings, compute the number of pairs resulting in a 'Pandigital Concatenation'.

Examples:

```
Input  : num[] = {"123567", "098234", "14765", "19804"}
Output : 3
The pairs, 1st and 2nd giving
(123567098234),1st and 4rd giving(12356719804) and
2nd and 3rd giving (09823414765),
on concatenation result in Pandigital Concatenations.

Input : num[] =  {"56789", "098345", "1234"}
Output : 0
None of the pairs on concatenation result in Pandigital
Concatenations.
```

Method 1 (Brute Force): A possible brute-force solution is to form all possible concatenations by forming all pairs in O(n^2 and using a frequency array for digits (0 – 9), we check if each digit exists at least once in each concatenation formed for every pair.

C++

```
 // C++ program to find all
// Pandigital concatenations
// of two strings.
```

```
#include <bits/stdc++.h>
using namespace std;

// Checks if a given
// string is Pandigital
bool isPanDigital(string s)
{
    bool digits[10] = {false};
    for (int i = 0; i < s.length(); i++)
        digits[s[i] - '0'] = true;

    // digit i is not present
    // thus not pandigital
    for (int i = 0; i <= 9; i++)
        if (digits[i] == false)
            return false;

    return true;
}

// Returns number of pairs
// of strings resulting in
// Pandigital Concatenations
int countPandigitalPairs(vector<string> &v)
{
    // iterate over all
    // pair of strings
    int pairs = 0;
    for (int i = 0; i < v.size(); i++)
        for (int j = i + 1; j < v.size(); j++)
            if (isPanDigital(v[i] + v[j]))
                pairs++;
    return pairs;
}

// Driver code
int main()
{
    vector<string> v = {"123567", "098234",
                        "14765", "19804"};
    cout << countPandigitalPairs(v) << endl;
    return 0;
}
```

Java

```
 // Java program to find all
// Pandigital concatenations
```

```
// of two strings.
import java.io.*;
import java.util.*;

class GFG
{
    static ArrayList<String> v =
                  new ArrayList<String>();

    // Checks if a given
    // string is Pandigital
    static int isPanDigital(String s)
    {
        int digits[] = new int[10];

        for (int i = 0; i < s.length(); i++)
            digits[s.charAt(i) -
                        (int)'0'] = 1;

        // digit i is not present
        // thus not pandigital
        for (int i = 0; i <= 9; i++)
            if (digits[i] == 0)
                return 0;

        return 1;
    }

    // Returns number of pairs
    // of strings resulting in
    // Pandigital Concatenations
    static int countPandigitalPairs()
    {
        // iterate over all
        // pair of strings
        int pairs = 0;
        for (int i = 0; i < v.size(); i++)
            for (int j = i + 1;
                     j < v.size(); j++)
                if (isPanDigital(v.get(i) +
                                 v.get(j)) == 1)
                    pairs++;
        return pairs;
    }

    // Driver code
    public static void main(String args[])
    {
```

```
        v.add("123567");
        v.add("098234");
        v.add("14765");
        v.add("19804");
        System.out.print(countPandigitalPairs());
    }
}

// This code is contributed
// by Manish Shaw(manishshaw1)
```

Python3

```
 # Python3 program to find all
# Pandigital concatenations
# of two strings.

# Checks if a given
# is Pandigital
def isPanDigital(s) :

    digits = [False] * 10;

    for i in range(0, len(s)) :
        digits[int(s[i]) -
               int('0')] = True

    # digit i is not present
    # thus not pandigital
    for i in range(0, 10) :
        if (digits[i] == False) :
            return False

    return True

# Returns number of pairs
# of strings resulting in
# Pandigital Concatenations
def countPandigitalPairs(v) :

    # iterate over all
    # pair of strings
    pairs = 0
    for i in range(0, len(v)) :

        for j in range (i + 1,
                        len(v)) :
```

```
            if (isPanDigital(v[i] +
                             v[j])) :
                pairs = pairs + 1
    return pairs

# Driver code
v = ["123567", "098234",
        "14765", "19804"]

print (countPandigitalPairs(v))

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# program to find all Pandigital
// concatenations of two strings.
using System;
using System.Collections.Generic;

class GFG
{
    // Checks if a given
    // string is Pandigital
    static int isPanDigital(string s)
    {
        int []digits = new int[10];
        Array.Clear(digits, 0, 10);
        for (int i = 0; i < s.Length; i++)
            digits[s[i] - (int)'0'] = 1;

        // digit i is not present
        // thus not pandigital
        for (int i = 0; i <= 9; i++)
            if (digits[i] == 0)
                return 0;

        return 1;
    }

    // Returns number of pairs
    // of strings resulting in
    // Pandigital Concatenations
    static int countPandigitalPairs(ref List<string> v)
    {
        // iterate over all
        // pair of strings
```

```
        int pairs = 0;
        for (int i = 0; i < v.Count; i++)
            for (int j = i + 1; j < v.Count; j++)
                if (isPanDigital(v[i] + v[j]) == 1)
                    pairs++;
        return pairs;
    }

    // Driver code
    static void Main()
    {
        List<string> v = new List<string>{"123567", "098234",
                                          "14765", "19804"};
        Console.WriteLine(countPandigitalPairs(ref v));
    }
}

// This code is contributed
// by Manish Shaw(manishshaw1)
```

PHP

```
 <?php
// PHP program to find all
// Pandigital concatenations
// of two strings.

// Checks if a given
// $is Pandigital
function isPanDigital($s)
{
    $digits = array();
    $digits = array_fill(0, 10, false);

    for ($i = 0; $i < strlen($s); $i++)
        $digits[ord($s[$i]) -
                ord('0')] = true;

    // digit i is not present
    // thus not pandigital
    for ($i = 0; $i <= 9; $i++)
        if ($digits[$i] == false)
            return false;

    return true;
}

// Returns number of pairs
```

```
// of strings resulting in
// Pandigital Concatenations
function countPandigitalPairs(&$v)
{
    // iterate over all
    // pair of strings
    $pairs = 0;
    for ($i = 0;
         $i < count($v); $i++)
    {
        for ($j = $i + 1;
             $j < count($v); $j++)
        {
            if (isPanDigital($v[$i].$v[$j]))
            {
                $pairs++;
            }
        }
    }
    return $pairs;
}

// Driver code
$v = array("123567", "098234",
           "14765", "19804");

echo (countPandigitalPairs($v));

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output:

```
3
```

Method 2 (Efficient):
Now we look for something better than the brute-force discussed above. Careful analysis suggests that, for every digit 0 – 9 to be present we have a mask as 1111111111 (i.e. all numbers 0-9 exist in the array of numbers

```
Digits -  0  1  2  3  4  5  6  7  8  9
          |  |  |  |  |  |  |  |  |  |
Mask   -  1  1  1  1  1  1  1  1  1  1
```

```
Here 1 denotes that the corresponding digits
exists at-least once thus for all such Pandigital
Concatenations, this relationship should hold.
So we can represent 11...11 as a valid mask for
pandigital concatenations.
```

So now the approach is to represent every string as a mask of 10 bits where the i^{th} bit is set if the i^{th} digit exists in the string.

```
E.g., "11405" can be represented as
Digits -           0  1  2  3  4  5  6  7  8  9
                   |  |  |  |  |  |  |  |  |  |
Mask for 11405 -   1  1  0  0  1  1  0  0  0  0
```

The approach though may look complete is still not efficient as we still have to iterate over all pairs and check if the OR of these two strings result in the mask of a valid Pandigital Concatenation.
If we analyze the possible masks of all possible strings we can understand that every single string would be only comprised of digits 0 – 9, so every number can at max contain all digits 0 to 9 at least once thus the mask of such a number would be 1111111111 (1023 in decimal). Thus in decimal system all masks exits in (0 – 1023].
Now we just have to maintain a frequency array to store the number of times a mask exists in the array of strings.

> Let two masks be i and j with frequencies $freq_i$ and $freq_j$ respectively,
> If (i OR j) = $Mask_{\text{pandigital concatenation}}$
> Then,
> Number of Pairs = $freq_i$ * $freq_j$

C++

```
 // CPP program to count PanDigital pairs
#include <bits/stdc++.h>
using namespace std;

const int pandigitalMask = ((1 << 10) - 1);

void computeMaskFrequencies(vector<string> v, map<int,
                                          int>& freq)
{
    for (int i = 0; i < v.size(); i++) {
        int mask = 0;

        // Stores digits present in string v[i]
        // atleast once. We use a set as we only
```

```
        // need digits which exist only once
        // (irrespective of reputation)
        unordered_set<int> digits;
        for (int j = 0; j < v[i].size(); j++)
            digits.insert(v[i][j] - '0');

        // Calculate the mask by considering all digits
        // existing atleast once
        for (auto it = digits.begin(); it != digits.end(); it++) {
            int digit = (*it);
            mask += (1 << digit);
        }

        // Increment the frequency of this mask
        freq[mask]++;
    }
}

// Returns number of pairs of strings resulting
// in Pandigital Concatenations
int pandigitalConcatenations(map<int, int> freq)
{
    int ans = 0;

    // All possible strings lie between 1 and 1023
    // so we iterate over every possible mask
    for (int i = 1; i <= 1023; i++) {
        for (int j = 1; j <= 1023; j++) {

            // if the concatenation results in mask of
            // Pandigital Concatenation, calculate all
            // pairs formed with Masks i and j
            if ((i | j) == pandigitalMask) {
                if (i == j)
                    ans += (freq[i] * (freq[i] - 1));
                else
                    ans += (freq[i] * freq[j]);
            }
        }
    }

    // since every pair is considers twice,
    // we get rid of half of these
    return ans/2;
}

int countPandigitalPairs(vector<string> v)
{
```

```
    // Find frequencies of all masks in
    // given vector of strings
    map<int, int> freq;
    computeMaskFrequencies(v, freq);

    // Return all possiblg concatenations.
    return pandigitalConcatenations(freq);
}

// Driver code
int main()
{
    vector<string> v = {"123567", "098234", "14765", "19804"};
    cout << countPandigitalPairs(v) << endl;
    return 0;
}
```

Output:

```
3
```

Complexity : O(N * |s| + 1023 * 1023) where |s| gives length of strings in the array

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/number-pairs-pandigital-concatenation/

Chapter 237

Number of unique triplets whose XOR is zero

Number of unique triplets whose XOR is zero - GeeksforGeeks

Given N numbers with no duplicates, count the number of unique triplets $(\mathbf{a_i}, \mathbf{a_j}, \mathbf{a_k})$ such that their XOR is 0. A triplet is said to be unique if all of the three numbers in the triplet is unique.

Examples:

```
Input : a[] = {1, 3, 5, 10, 14, 15};
Output : 2
Explanation : {1, 14, 15} and {5, 10, 15} are the
              unique triplets whose XOR is 0.
              {1, 14, 15} and all other combinations of
              1, 14, 15 are considered as 1 only.

Input : a[] = {4, 7, 5, 8, 3, 9};
Output : 1
Explanation : {4, 7, 3} is the only triplet whose XOR is 0
```

Naive Approach: A naive approach is to run three nested loops, the first runs from 0 to n, second from i+1 to n, and the last one from j+1 to n to get the unique triplets. Calculate the XOR of a_i, a_j, a_k, check if it equals to 0, if so, then increase the count.
Time Complexity : $O(n^3)$

Efficient Approach: An efficient approach is to use one of the properties of XOR that XOR of two same numbers gives 0. So we need to calculate XOR of unique pairs only, and if the calculated XOR is one of the array element, then we get the triplet whose XOR is 0. Given below are the steps for counting the number of unique triplets:

Below is the complete algorithm of this approach:

1. With map, mark all the array elements.
2. Run two nested loops, one from i-n, and the other from i+1-n to get all the pairs.
3. Obtain the XOR of pair.
4. Check if the XOR is an array element and not one of a_i or a_j.
5. Increase the count if the condition holds.
6. Return count/3 as we only want unique triplets. Since i-n and j+1-n gives us unique pairs but not triplets, so we do a count/3 to remove the other two possible combinations.

Below is the implementation of above idea:

C++

```
 // CPP program to count the number of
// unique triplets whose XOR is 0
#include <bits/stdc++.h>
using namespace std;

// function to count the number of
// unique triplets whose xor is 0
int countTriplets(int a[], int n)
{
    // To store values that are present
    unordered_set<int> s;
    for (int i = 0; i < n; i++)
        s.insert(a[i]);

    // stores the count of unique triplets
    int count = 0;

    // traverse for all i, j pairs such that j>i
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {

          // xor of a[i] and a[j]
          int xr = a[i] ^ a[j];

          // if xr of two numbers is present,
          // then increase the count
          if (s.find(xr) != s.end() && xr != a[i] &&
                                       xr != a[j])
            count++;
        }
    }

    // returns answer
    return count / 3;
```

```
}

// Driver code to test above function
int main()
{
    int a[] = {1, 3, 5, 10, 14, 15};
    int n = sizeof(a) / sizeof(a[0]);
    cout << countTriplets(a, n);
    return 0;
}
```

Java

```
 // Java program to count
// the number of unique
// triplets whose XOR is 0
import java.io.*;
import java.util.*;

class GFG
{
    // function to count the
    // number of unique triplets
    // whose xor is 0
    static int countTriplets(int []a,
                             int n)
    {
        // To store values
        // that are present
        ArrayList<Integer> s =
                  new ArrayList<Integer>();
        for (int i = 0; i < n; i++)
            s.add(a[i]);

        // stores the count
        // of unique triplets
        int count = 0;

        // traverse for all i,
        // j pairs such that j>i
        for (int i = 0; i < n; i++)
        {
            for (int j = i + 1;
                     j < n; j++)
            {

            // xor of a[i] and a[j]
            int xr = a[i] ^ a[j];
```

```
            // if xr of two numbers
            // is present, then
            // increase the count
            if (s.contains(xr) &&
                xr != a[i] && xr != a[j])
                count++;
            }
        }

        // returns answer
        return count / 3;
    }

    // Driver code
    public static void main(String srgs[])
    {
        int []a = {1, 3, 5,
                   10, 14, 15};
        int n = a.length;
        System.out.print(countTriplets(a, n));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

C#

```
 // C# program to count
// the number of unique
// triplets whose XOR is 0
using System;
using System.Collections.Generic;

class GFG
{
    // function to count the
    // number of unique triplets
    // whose xor is 0
    static int countTriplets(int []a,
                             int n)
    {
        // To store values
        // that are present
        List<int> s = new List<int>();
        for (int i = 0; i < n; i++)
            s.Add(a[i]);
```

```
        // stores the count
        // of unique triplets
        int count = 0;

        // traverse for all i,
        // j pairs such that j>i
        for (int i = 0; i < n; i++)
        {
            for (int j = i + 1;
                     j < n; j++)
            {

            // xor of a[i] and a[j]
            int xr = a[i] ^ a[j];

            // if xr of two numbers
            // is present, then
            // increase the count
            if (s.Exists(item => item == xr) &&
                   xr != a[i] && xr != a[j])
                count++;
            }
        }

        // returns answer
        return count / 3;
    }

    // Driver code
    static void Main()
    {
        int []a = new int[]{1, 3, 5,
                            10, 14, 15};
        int n = a.Length;
        Console.Write(countTriplets(a, n));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

```
2
```

Time Complexity : $O(n^2)$

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/number-unique-triplets-whose-xor-zero/

Chapter 238

Number whose XOR sum with given array is a given number k

Number whose XOR sum with given array is a given number k - GeeksforGeeks

Given an array of N numbers and a number K. The task is to insert a number in the given array such that the bitwise XOR of all the elements in the new array equals the given input K.

Examples:

```
Input:
a = {1, 2, 3, 4, 5}, k = 10
Output: 11
Explanation: 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 11 = 10

Input: A[] = { 12, 23, 34, 56, 78 }, k = 6
Output: 73
```

Approach: The basic idea is to use the simple XOR property, i.e. if X ⌃ Y = Z then X ⌃ Z = Y. Let's suppose the number to be inserted in array be X such that (A[0] ⌃ A[1] ⌃ … ⌃ A[n – 1]) ⌃ X = k. Thus, to find X we can use the relation (A[0] ⌃ A[1] ⌃ … ⌃ A[n – 1]) ⌃ k = X.

Below is the implementation of above approach.

```
 // CPP Program to find the number
// whose XOR sum with given array is
// equal to a given number k
#include <bits/stdc++.h>
using namespace std;
```

```
// This function returns the number to
// be inserted in the given array
int findEletobeInserted(int A[], int n, int k)
{
    // initialise the answer with k
    int ans = k;
    for (int i = 0; i < n; i++)
        ans ^= A[i]; // XOR of all elements in the array
    return ans;
}

// Driver Code to test above function
int main()
{
    int A[] = { 1, 2, 3, 4, 5 };
    int n = sizeof(A) / sizeof(A[0]);
    int k = 10;

    cout << findEletobeInserted(A, n, k)
         << " has to be inserted"
         " in the given array to make xor sum of "
         << k << endl;

    return 0;
}
```

Java

```
 // Java Program to find the number
// whose XOR sum with given array is
// equal to a given number k
import java.io.*;

class GFG {

    // This function returns the number to
    // be inserted in the given array
    static int findEletobeInserted(int A[],
                                   int n, int k)
    {

        // initialise the answer with k
        int ans = k;
        for (int i = 0; i < n; i++)

            // XOR of all elements in
            // the array
            ans ^= A[i];
```

```
        return ans;
    }

    // Driver Code to test above function
    public static void main (String[] args)
    {
        int A[] = { 1, 2, 3, 4, 5 };
        int n =A.length;
        int k = 10;

        System.out.println(
             findEletobeInserted(A, n, k)
              + " has to be inserted in "
              + "the given array to make"
                   + " xor sum of " + k);
    }
}

// This code is contributed by anuj_67.
```

C#

```
 // C# Program to find the number
// whose XOR sum with given array is
// equal to a given number k
using System ;

class GFG {

    // This function returns the number to
    // be inserted in the given array
    static int findEletobeInserted(int []A,
                            int n, int k)
    {

        // initialise the answer with k
        int ans = k;
        for (int i = 0; i < n; i++)

            // XOR of all elements in
            // the array
            ans ^= A[i];
        return ans;
    }

    // Driver Code to test above function
    public static void Main ()
    {
```

```
        int []A = { 1, 2, 3, 4, 5 };
        int n =A.Length;
        int k = 10;

        Console.WriteLine(
            findEletobeInserted(A, n, k)
            + " has to be inserted in "
            + "the given array to make"
                + " xor sum of " + k);
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP Program to find the number
// whose XOR sum with given array is
// equal to a given number k

// This function returns the number to
// be inserted in the given array
function findEletobeInserted($A, $n, $k)
{
    // initialise the answer with k
    $ans = $k;
    for ( $i = 0; $i < $n; $i++)

        // XOR of all elements
        // in the array
        $ans ^= $A[$i];
    return $ans;
}

    // Driver Code
    $A = array(1, 2, 3, 4, 5);
    $n = count($A);
    $k = 10;

    echo findEletobeInserted($A, $n, $k) ;
    echo " has to be inserted";
    echo " in the given array to make xor sum of ";
    echo $k , "\n";

// This code is contributed by anuj_67.
?>
```

Output :

```
11 has to be inserted in the given array to make xor sum of 10
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/number-whose-xor-sum-given-array-given-number-k/

Chapter 239

Number whose sum of XOR with given array range is maximum

Number whose sum of XOR with given array range is maximum - GeeksforGeeks

You are given a sequence of N integers and Q queries. In each query, you are given two parameters L and R. You have to find the smallest integer X such that 0 <= X < 2^31 and the sum of XOR of x with all elements is range [L, R] is maximum possible.

Examples :

```
Input  : A = {20, 11, 18, 2, 13}
         Three queries as (L, R) pairs
         1 3
         3 5
         2 4
Output : 2147483629
         2147483645
         2147483645
```

Approach: The binary representation of each element and X, we can observe that each bit is independent and the problem can be solved by iterating over each bit. Now basically for each bit we need to count the number of 1's and 0's in the given range, if the number of 1's are more then you have to set that bit of X to 0 so that the sum is maximum after xor with X else if number of 0's are more then you have to set that bit of X to 1. If the number of 1's and 0's are equal then we can set that bit of X to any one of 1 or 0 because it will not affect the sum, but we have to minimize the value of X so we will take that bit 0.

Now, to optimize the solution we can pre-calculate the count of 1's at each bit position of the numbers up to that position by making a prefix array this will take O(n) time. Now for

each query number of 1's will be the number of 1's up to Rth position – number of 1's up to (L-1)th position.

C++

```
 // CPP program to find smallest integer X
// such that sum of its XOR with range is
// maximum.
#include <bits/stdc++.h>
using namespace std;

#define MAX 2147483647
int one[100001][32];

// Function to make prefix array which
// counts 1's of each bit up to that number
void make_prefix(int A[], int n)
{
    for (int j = 0; j < 32; j++)
        one[0][j] = 0;

    // Making a prefix array which sums
    // number of 1's up to that position
    for (int i = 1; i <= n; i++)
    {
        int a = A[i - 1];
        for (int j = 0; j < 32; j++)
        {
            int x = pow(2, j);

            // If j-th bit of a number is set then
            // add one to previously counted 1's
            if (a & x)
                one[i][j] = 1 + one[i - 1][j];
            else
                one[i][j] = one[i - 1][j];
        }
    }
}

// Function to find X
int Solve(int L, int R)
{
    int l = L, r = R;
    int tot_bits = r - l + 1;

    // Initially taking maximum value all bits 1
    int X = MAX;
```

```
    // Iterating over each bit
    for (int i = 0; i < 31; i++)
    {

        // get 1's at ith bit between the
        // range L-R by subtracting 1's till
        // Rth number - 1's till L-1th number
        int x = one[r][i] - one[l - 1][i];

        // If 1's are more than or equal to 0's
        // then unset the ith bit from answer
        if (x >= tot_bits - x)
        {
            int ith_bit = pow(2, i);

            // Set ith bit to 0 by doing
            // Xor with 1
            X = X ^ ith_bit;
        }
    }
    return X;
}

// Driver program
int main()
{
    // Taking inputs
    int n = 5, q = 3;
    int A[] = { 210, 11, 48, 22, 133 };
    int L[] = { 1, 4, 2 }, R[] = { 3, 14, 4 };

    make_prefix(A, n);

    for (int j = 0; j < q; j++)
        cout << Solve(L[j], R[j]) << endl;

    return 0;
}
```

Java

```
 // Java program to find smallest integer X
// such that sum of its XOR with range is
// maximum.
import java.lang.Math;

class GFG {
```

```
private static final int MAX = 2147483647;
static int[][] one = new int[100001][32];

// Function to make prefix array which counts
// 1's of each bit up to that number
static void make_prefix(int A[], int n)
{
    for (int j = 0; j < 32; j++)
        one[0][j] = 0;

    // Making a prefix array which sums
    // number of 1's up to that position
    for (int i = 1; i <= n; i++)
    {
        int a = A[i - 1];
        for (int j = 0; j < 32; j++)
        {
            int x = (int)Math.pow(2, j);

            // If j-th bit of a number is set then
            // add one to previously counted 1's
            if ((a & x) != 0)
                one[i][j] = 1 + one[i - 1][j];
            else
                one[i][j] = one[i - 1][j];
        }
    }
}

// Function to find X
static int Solve(int L, int R)
{
    int l = L, r = R;
    int tot_bits = r - l + 1;

    // Initially taking maximum
    // value all bits 1
    int X = MAX;

    // Iterating over each bit
    for (int i = 0; i < 31; i++)
    {

        // get 1's at ith bit between the range
        // L-R by subtracting 1's till
        // Rth number - 1's till L-1th number
        int x = one[r][i] - one[l - 1][i];
```

```
            // If 1's are more than or equal to 0's
            // then unset the ith bit from answer
            if (x >= tot_bits - x)
            {
                int ith_bit = (int)Math.pow(2, i);

                // Set ith bit to 0 by
                // doing Xor with 1
                X = X ^ ith_bit;
            }
        }
        return X;
    }

    // Driver program
    public static void main(String[] args)
    {
        // Taking inputs
        int n = 5, q = 3;
        int A[] = { 210, 11, 48, 22, 133 };
        int L[] = { 1, 4, 2 }, R[] = { 3, 14, 4 };

        make_prefix(A, n);

        for (int j = 0; j < q; j++)
            System.out.println(Solve(L[j], R[j]));
    }
}

// This code is contributed by Smitha
```

Python3

```
 # Python3 program to find smallest integer X
# such that sum of its XOR with range is
# maximum.
import math

one = [[0 for x in range(32)]
      for y in range(100001)]
MAX = 2147483647

# Function to make prefix array
# which counts 1's of each bit
# up to that number
def make_prefix(A, n) :
    global one, MAX
```

```
    for j in range(0 , 32) :
        one[0][j] = 0

    # Making a prefix array which
    # sums number of 1's up to
    # that position
    for i in range(1, n+1) :
        a = A[i - 1]
        for j in range(0 , 32) :

            x = int(math.pow(2, j))

            # If j-th bit of a number
            # is set then add one to
            # previously counted 1's
            if (a & x) :
                one[i][j] = 1 + one[i - 1][j]
            else :
                one[i][j] = one[i - 1][j]

# Function to find X
def Solve(L, R) :

    global one, MAX
    l = L
    r = R
    tot_bits = r - l + 1

    # Initially taking maximum
    # value all bits 1
    X = MAX

    # Iterating over each bit
    for i in range(0, 31) :

        # get 1's at ith bit between the
        # range L-R by subtracting 1's till
        # Rth number - 1's till L-1th number

        x = one[r][i] - one[l - 1][i]

        # If 1's are more than or equal
        # to 0's then unset the ith bit
        # from answer
        if (x >= (tot_bits - x)) :

            ith_bit = pow(2, i)
```

```
            # Set ith bit to 0 by
            # doing Xor with 1
            X = X ^ ith_bit
    return X

# Driver Code
n = 5
q = 3
A = [ 210, 11, 48, 22, 133 ]
L = [ 1, 4, 2 ]
R = [ 3, 14, 4 ]

make_prefix(A, n)

for j in range(0, q) :
    print (Solve(L[j], R[j]),end="\n")

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# program to find smallest integer X
// such that sum of its XOR with range is
// maximum.
using System;
using System.Collections.Generic;

class GFG{
    static int MAX = 2147483647;
    static int [,]one = new int[100001,32];

    // Function to make prefix
    // array which counts 1's
    // of each bit up to that number
    static void make_prefix(int []A, int n)
    {
        for (int j = 0; j < 32; j++)
            one[0,j] = 0;

        // Making a prefix array which sums
        // number of 1's up to that position
        for (int i = 1; i <= n; i++)
        {
            int a = A[i - 1];
            for (int j = 0; j < 32; j++)
            {
                int x = (int)Math.Pow(2, j);
```

```
            // If j-th bit of a number is set then
            // add one to previously counted 1's
            if ((a & x) != 0)
                one[i, j] = 1 + one[i - 1, j];
            else
                one[i,j] = one[i - 1, j];
        }
    }
}

// Function to find X
static int Solve(int L, int R)
{
    int l = L, r = R;
    int tot_bits = r - l + 1;

    // Initially taking maximum
    // value all bits 1
    int X = MAX;

    // Iterating over each bit
    for (int i = 0; i < 31; i++)
    {

        // get 1's at ith bit between the
        // range L-R by subtracting 1's till
        // Rth number - 1's till L-1th number
        int x = one[r, i] - one[l - 1, i];

        // If 1's are more than or
        // equal to 0's then unset
        // the ith bit from answer
        if (x >= tot_bits - x)
        {
            int ith_bit = (int)Math.Pow(2, i);

            // Set ith bit to 0 by doing
            // Xor with 1
            X = X ^ ith_bit;
        }
    }
    return X;
}

// Driver Code
public static void Main()
{
```

```
        // Taking inputs
        int n = 5, q = 3;
        int []A = {210, 11, 48, 22, 133};
        int []L = {1, 4, 2};
        int []R = {3, 14, 4};

        make_prefix(A, n);

        for (int j = 0; j < q; j++)
            Console.WriteLine(Solve(L[j], R[j]));
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

PHP

```
 <?php
error_reporting(0);
// PHP program to find smallest integer X
// such that sum of its XOR with range is
// maximum.

$one = array();
$MAX = 2147483647;

// Function to make prefix array
// which counts 1's of each bit
// up to that number
function make_prefix($A, $n)
{
    global $one, $MAX;

    for ($j = 0; $j < 32; $j++)
        $one[0][$j] = 0;

    // Making a prefix array which
    // sums number of 1's up to
    // that position
    for ($i = 1; $i <= $n; $i++)
    {
        $a = $A[$i - 1];
        for ($j = 0; $j < 32; $j++)
        {
            $x = pow(2, $j);
```

```
            // If j-th bit of a number
            // is set then add one to
            // previously counted 1's
            if ($a & $x)
                $one[$i][$j] = 1 + $one[$i - 1][$j];
            else
                $one[$i][$j] = $one[$i - 1][$j];
        }
    }
}

// Function to find X
function Solve($L, $R)
{
    global $one, $MAX;
    $l = $L; $r = $R;
    $tot_bits = $r - $l + 1;

    // Initially taking maximum
    // value all bits 1
    $X = $MAX;

    // Iterating over each bit
    for ($i = 0; $i < 31; $i++)
    {
        // get 1's at ith bit between the
        // range L-R by subtracting 1's till
        // Rth number - 1's till L-1th number

        $x = $one[$r][$i] - $one[$l - 1][$i];

        // If 1's are more than or equal
        // to 0's then unset the ith bit
        // from answer
        if ($x >= ($tot_bits - $x))
        {
            $ith_bit = pow(2, $i);

            // Set ith bit to 0 by
            // doing Xor with 1
            $X = $X ^ $ith_bit;
        }
    }
    return $X;
}

// Driver Code
$n = 5; $q = 3;
```

```
$A = [ 210, 11, 48, 22, 133 ];
$L = [ 1, 4, 2 ];
$R = [ 3, 14, 4 ];

make_prefix($A, $n);

for ($j = 0; $j < $q; $j++)
    echo (Solve($L[$j], $R[$j]). "\n");

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output :

```
2147483629
2147483647
2147483629
```

Improved By : Smitha Dinesh Semwal, manishshaw1

Source

https://www.geeksforgeeks.org/number-whose-sum-of-xor-with-given-array-range-is-maximum/

Chapter 240

Number with set bits only between L-th and R-th index

Number with set bits only between L-th and R-th index - GeeksforGeeks

Given L and R. The task is to find the number in whose binary representation all bits between the L-th and R-th index are set and the rest of the bits are unset. The binary representation is of 32 bits.

Examples:

> **Input:** L = 2, R = 5
> **Output:** 60
> **Explanation:** The binary representation is
> 0..0111100 => 60
>
> **Input:** L = 1, R = 3
> **Output:** 14
> **Explanation:** The binary representation is
> 0..01110 => 14

Naive Approach: The naive approach to find the number is to iterate from i = L to i = R and calculate the addition of all the powers of 2^i.

Below program illustrate the naive approach:

C++

```
 // CPP program to print the integer
// with all the bits set in range L-R
// Naive Approach
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to return the integer
// with all the bits set in range L-R
int getInteger(int L, int R)
{

    int number = 0;

    // iterate from L to R
    // and add all powers of 2
    for (int i = L; i <= R; i++)
        number += pow(2, i);

    return number;
}

// Driver Code
int main()
{
    int L = 2, R = 5;
    cout << getInteger(L, R);
    return 0;
}
```

Java

```
 // Java program to print the
// integer with all the bits
// set in range L-R Naive Approach
import java.io.*;

class GFG
{

// Function to return the
// integer with all the
// bits set in range L-R
static int getInteger(int L,
                      int R)
{
    int number = 0;

    // iterate from L to R
    // and add all powers of 2
    for (int i = L; i <= R; i++)
        number += Math.pow(2, i);

    return number;
```

```
}

// Driver Code
public static void main (String[] args)
{
    int L = 2, R = 5;
    System.out.println(getInteger(L, R));
}
}

// This code is contributed by anuj_67..
```

C#

```
 // C# program to print the
// integer with all the bits
// set in range L-R Naive Approach
using System;

class GFG
{
// Function to return the
// integer with all the
// bits set in range L-R
static int getInteger(int L,
                      int R)
{
    int number = 0;

    // iterate from L to R
    // and add all powers of 2
    for (int i = L; i <= R; i++)
        number += (int)Math.Pow(2, i);

    return number;
}

// Driver Code
public static void Main ()
{
    int L = 2, R = 5;
    Console.Write(getInteger(L, R));
}
}

// This code is contributed
// by shiv_bhakt.
```

PHP

```
 <?php
// PHP program to print
// the integer with all
// the bits set in range
// L-R Naive Approach

// Function to return the
// integer with all the
// bits set in range L-R
function getInteger($L, $R)
{
    $number = 0;

    // iterate from L to R
    // and add all powers of 2
    for ($i = $L; $i <= $R; $i++)
        $number += pow(2, $i);

    return $number;
}

// Driver Code
$L = 2;
$R = 5;
echo getInteger($L, $R);

// This code is contributed
// by shiv_bhakt.
?>
```

Output:

```
60
```

An **efficient approach** is to compute the number with all (R) set bits from right and subtract the number with all (L-1) bits set from right to get the required number.

1. Compute the number which has all R set bits from the right using the below formula.

   `(1 << (R+1)) - 1.`

2. Subtract the number which has all (L-1) set bits from the right.

   `(1<<L) - 1`

Hence computing ((1<<(R+1))-1)-((1<<L)-1), we get the final formula as:

(1<<(R+1))-(1<<L)

Below program illustrate the efficient approach:

C++

```
 // CPP program to print the integer
// with all the bits set in range L-R
// Efficient Approach
#include <bits/stdc++.h>
using namespace std;

// Function to return the integer
// with all the bits set in range L-R
int setbitsfromLtoR(int L, int R)
{
    return (1 << (R + 1)) - (1 << L);
}

// Driver Code
int main()
{
    int L = 2, R = 5;
    cout << setbitsfromLtoR(L, R);
    return 0;
}
```

Java

```
 // Java program to print
// the integer with all
// the bits set in range
// L-R Efficient Approach
import java.io.*;

class GFG
{

// Function to return the
// integer with all the
// bits set in range L-R
static int setbitsfromLtoR(int L,
                           int R)
{
```

```
    return (1 << (R + 1)) -
           (1 << L);
}

// Driver Code
public static void main (String[] args)
{
    int L = 2, R = 5;
    System.out.println(setbitsfromLtoR(L, R));
}
}

// This code is contributed
// by shiv_bhakt.
```

Python3

```
 # Python3 program to print
# the integer with all the
# bits set in range L-R
# Efficient Approach

# Function to return the
# integer with all the
# bits set in range L-R
def setbitsfromLtoR(L, R):

    return ((1 << (R + 1)) -
            (1 << L))

# Driver Code
L = 2
R = 5
print(setbitsfromLtoR(L, R))

# This code is contributed
# by Smita
```

C#

```
 // C# program to print
// the integer with all
// the bits set in range
// L-R Efficient Approach
using System;

class GFG
```

```
{
// Function to return the
// integer with all the
// bits set in range L-R
static int setbitsfromLtoR(int L,
                           int R)
{
    return (1 << (R + 1)) -
           (1 << L);
}

// Driver Code
public static void Main ()
{
    int L = 2, R = 5;
    Console.WriteLine(setbitsfromLtoR(L, R));
}
}

// This code is contributed
// by shiv_bhakt.
```

PHP

```
 <?php
// PHP program to print
// the integer with all
// the bits set in range
// L-R Efficient Approach

// Function to return the
// integer with all the
// bits set in range L-R
function setbitsfromLtoR($L, $R)
{
    return (1 << ($R + 1)) -
           (1 << $L);
}

// Driver Code
$L = 2;
$R = 5;
echo setbitsfromLtoR($L, $R);

// This code is contributed
// by shiv_bhakt.
?>
```

Output:

```
60
```

Time Complexity: O(1)
Auxiliary Space: O(1)

Improved By : vt_m, shiv_bhakt, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/number-with-set-bits-only-between-l-th-and-r-th-index/

Chapter 241

Numbers whose bitwise OR and sum with N are equal

Numbers whose bitwise OR and sum with N are equal - GeeksforGeeks

Given a non-negative integer N, the task is to find count of non-negative integers less than or equal to N whose bitwise OR and sum with N are equal.
Examples :

```
Input  : N = 3
Output : 1
0 is the only number in [0, 3]
that satisfies given property.
(0 + 3) = (0 | 3)

Input  : 10
Output : 4
(0 + 10) = (0 | 10) (Both are 10)
(1 + 10) = (1 | 10) (Both are 11)
(4 + 10) = (4 | 10) (Both are 14)
(5 + 10) = (5 | 10) (Both are 15)
```

A **simple solution** is to traverse all numbers from 0 to N and do bitwise OR and SUM with N, if both are equal increment counter.
Time complexity = O(N).

An **efficient solution** is to follow following steps.
1. Find count of zero bit in N.
2. Return pow(2,count).

The idea is based on the fact that bitwise OR and sum of a number x with N are equal, if and only if
bitwise AND of x with N will is 0

```
Let, N=10 =10102.
Bitwise AND of a number with N will be 0, if number
contains zero bit with all respective set bit(s) of
N and either zero bit or set bit with all respective
zero bit(s) of N (because, 0&0=1&0=0).
  e.g.
bit     : 1   0   1   0
position: 4   3   2   1
Bitwise AND of any number with N will be 0, if the number
has following bit pattern
1st position can be  either 0 or 1 (2 ways)
2nd position can be  1 (1 way)
3rd position can be  either 0 or 1 (2 ways)
4th position can be  1 (1 way)
Total count = 2*1*2*1 = 22 = 4.
```

C++

```
 // C++ program to count numbers whose bitwise
// OR and sum with N are equal
#include <bits/stdc++.h>
using namespace std;

// Function to find total 0 bit in a number
unsigned int CountZeroBit(int n)
{
    unsigned int count = 0;
    while(n)
    {
       if (!(n & 1))
           count++;
       n >>= 1;
    }
    return count;
}

// Function to find Count of non-negative numbers
// less than or equal to N, whose bitwise OR and
// SUM with N are equal.
int CountORandSumEqual(int N )
{
    // count number of zero bit in N
    int count = CountZeroBit(N);

    // power of 2 to count
    return (1 << count);
```

```
}

// Driver code
int main()
{
   int N = 10;
   cout << CountORandSumEqual(N);
   return 0;
}
```

Java

```
 // Java program to count numbers whose bitwise
// OR and sum with N are equal
class GFG {

    // Function to find total 0 bit in a number
    static int CountZeroBit(int n)
    {
        int count = 0;

        while(n > 0)
        {
            if ((n & 1) != 0)
                count++;

            n >>= 1;
        }

        return count;
    }

    // Function to find Count of non-negative
    // numbers less than or equal to N, whose
    // bitwise OR and SUM with N are equal.
    static int CountORandSumEqual(int N )
    {

        // count number of zero bit in N
        int count = CountZeroBit(N);

        // power of 2 to count
        return (1 << count);
    }

    //Driver code
    public static void main (String[] args)
    {
```

```
        int N = 10;

        System.out.print(CountORandSumEqual(N));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to count numbers whose
# bitwise OR and sum with N are equal

# Function to find total 0 bit in a number
def CountZeroBit(n):

    count = 0
    while(n):

        if (not(n & 1)):
            count += 1
        n >>= 1

    return count

# Function to find Count of non-negative
# numbers less than or equal to N, whose
# bitwise OR and SUM with N are equal.
def CountORandSumEqual(N):

    # count number of zero bit in N
    count = CountZeroBit(N)

    # power of 2 to count
    return (1 << count)

# Driver code
N = 10
print(CountORandSumEqual(N))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to count numbers whose
// bitwise OR and sum with N are equal
```

```
using System;

class GFG
{

    // Function to find total
    // 0 bit in a number
    static int CountZeroBit(int n)
    {
        int count = 0;
        while(n>0)
        {
            if (n%2!=0)
            count++;
            n >>= 1;
        }
        return count;
    }

    // Function to find Count of non-negative
    // numbers less than or equal to N, whose
    // bitwise OR and SUM with N are equal.
    static int CountORandSumEqual(int N )
    {

    // count number of zero bit in N
    int count = CountZeroBit(N);

    // power of 2 to count
    return (1 << count);
    }

    //Driver code
    public static void Main()
    {
        int N = 10;
        Console.Write(CountORandSumEqual(N));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program to count
// numbers whose bitwise
// OR and sum with N are equal
```

```
// Function to find total
// 0 bit in a number

function CountZeroBit($n)
{
    $count = 0;
    while($n)
    {
    if (!($n & 1))
        $count++;
    $n >>= 1;
    }
    return $count;
}

// Function to find Count of
// non-negative numbers less
// than or equal to N, whose
// bitwise OR and SUM with N
// are equal.

function CountORandSumEqual($N )
{
    // count number of
    // zero bit in N
    $count = CountZeroBit($N);

    // power of 2 to count
    return (1 << $count);
}

// Driver code
$N = 10;
echo CountORandSumEqual($N);

// This code is contributed by Ajit
?>
```

Output :

```
 4
```

Total time complexity of above solution will be $O(\log_2(N))$.

Improved By : jit_t

Source

https://www.geeksforgeeks.org/numbers-whose-bitwise-sum-n-equal/

Chapter 242

Odd numbers in N-th row of Pascal's Triangle

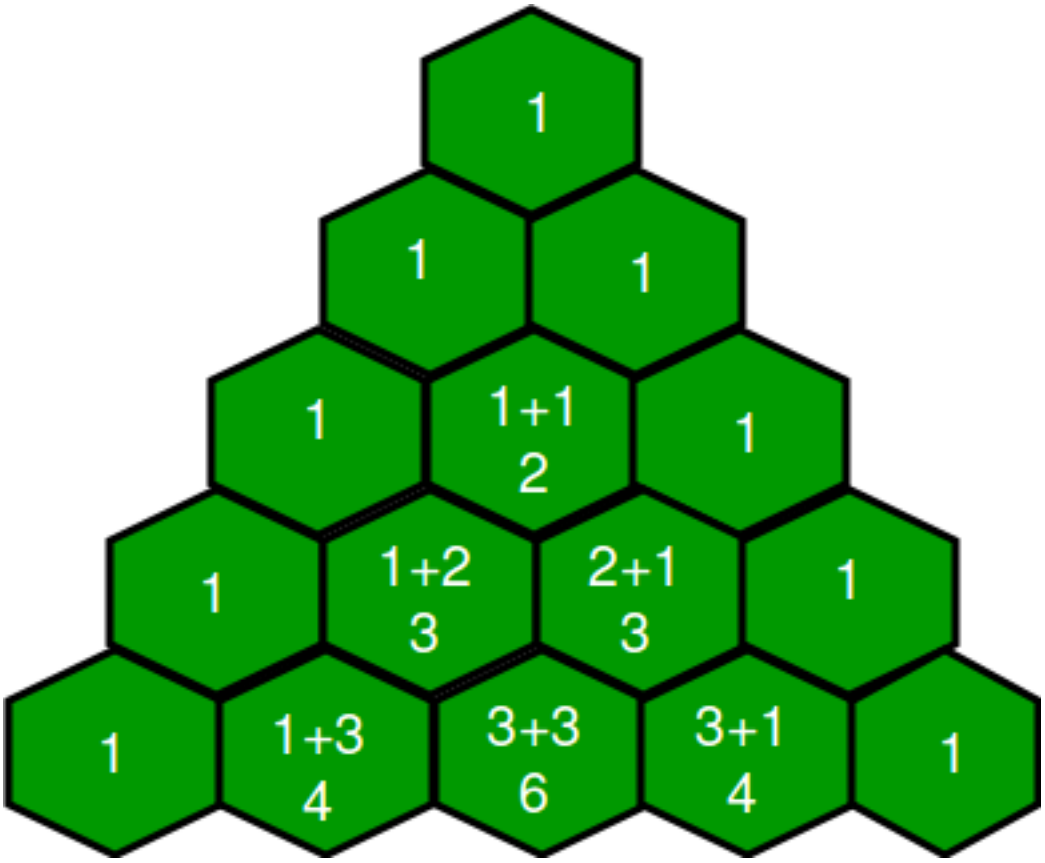
Odd numbers in N-th row of Pascal's Triangle - GeeksforGeeks

Given N, the row number of Pascal's triangle(row starting from 0). Find the count of odd numbers in N-th row of Pascal's Triangle.

Prerequisite : Pascal's Triangle | Count number of 1's in binary representation of N

Examples :

```
Input : 11
Output : 8

Input : 20
Output : 4
```



Approach : It appears the answer is always a power of 2. In fact, the following theorem exists :
THEOREM : The number of odd entries in row N of Pascal's Triangle is 2 raised to the number of 1's in the binary expansion of N.
Example: Since 83 = 64 + 16 + 2 + 1 has binary expansion (1010011), then row 83 has pow(2, 4) = 16 odd numbers.

Below is the implementation of above approach :

C++

```
// CPP code to find the count of odd numbers
// in n-th row of Pascal's Triangle
#include <bits/stdc++.h>
using namespace std ;

/* Function to get no of set
   bits in binary representation
   of positive integer n */
int countSetBits(int n)
```

```
{
    unsigned int count = 0;
    while (n)
    {
        count += n & 1;
        n >>= 1;
    }

    return count;
}

int countOfOddsPascal(int n)
{
    // Count number of 1's in binary
    // representation of n.
    int c = countSetBits(n);

    // Number of odd numbers in n-th
    // row is 2 raised to power the count.
    return pow(2, c);
}

// Driver code
int main()
{
    int n = 20;
    cout << countOfOddsPascal(n) ;
    return 0;
}
```

Java

```
 // Java code to find the count of odd
// numbers in n-th row of Pascal's
// Triangle
import java.io.*;

class GFG {

    /* Function to get no of set
    bits in binary representation
    of positive integer n */
    static int countSetBits(int n)
    {
        long count = 0;
        while (n > 0)
        {
            count += n & 1;
```

```
            n >>= 1;
        }

        return (int)count;
    }

    static int countOfOddsPascal(int n)
    {

        // Count number of 1's in binary
        // representation of n.
        int c = countSetBits(n);

        // Number of odd numbers in n-th
        // row is 2 raised to power the
        // count.
        return (int)Math.pow(2, c);
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 20;
        System.out.println(
                    countOfOddsPascal(n));
    }
}

// This code is contributed by anuj_67.
```

Python3

```
 # Python code to find the count of
# odd numbers in n-th row of
# Pascal's Triangle

# 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

def countOfOddPascal(n):
```

```
    # Count number of 1's in binary
    # representation of n.
    c = countSetBits(n)

    # Number of odd numbers in n-th
    # row is 2 raised to power the count.
    return pow(2, c)

# Driver Program
n = 20
print(countOfOddPascal(n))

# This code is contributed by Shrikant13
```

C#

```
 // C# code to find the count of odd numbers
// in n-th row of Pascal's Triangle
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;
    }

    static int countOfOddsPascal(int n)
    {
        // Count number of 1's in binary
        // representation of n.
        int c = countSetBits(n);

        // Number of odd numbers in n-th
        // row is 2 raised to power the
        // count.
        return (int)Math.Pow(2, c);
```

```
    }

    // Driver code
    public static void Main ()
    {
        int n = 20;
        Console.WriteLine(
                 countOfOddsPascal(n)) ;
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP code to find the
// count of odd numbers
// in n-th row of Pascal's
// Triangle

/* 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;
}

function countOfOddsPascal($n)
{

    // Count number of 1's in binary
    // representation of n.
    $c = countSetBits($n);

    // Number of odd numbers in n-th
    // row is 2 raised to power the count.
    return pow(2, $c);
}
```

```
    // Driver code
    $n = 20;
    echo countOfOddsPascal($n) ;

// This code is contributed by mits.
?>
```

Output:

```
4
```

Time Complexity : O(L), where L is the length of binary representation of given N.
Reference : https://www.math.hmc.edu/funfacts/ffiles/30001.4-5.shtml

Improved By : shrikanth13, Mithun Kumar, vt_m

Source

https://www.geeksforgeeks.org/odd-numbers-in-n-th-row-of-pascals-triangle/

Chapter 243

Odious number

Odious number - GeeksforGeeks

Odious number is a nonnegative number that has an odd number of 1s in its binary expansion. The first few odious numbers are therefore 1, 2, 4, 7, 8, 11, 13, 14, 16, 19...

Given a number check if its a odious number or not.

Examples :

```
Input : 16
Output : Odious Number
Explanation: Binary expansion of 16 = 10000,
having number of 1s =1 i.e odd.

Input : 23
Output :  Not odious number
Explanation: Binary expansion of 23 is 10111,
the number of 1s in this is 4 i.e even.
```

1) Count set bits in given number.
2) Return true if the count is odd, false otherwise.

C++

```
 // C/C++ program to check if a number is
// Odious Number or not
#include <iostream>
using namespace std;
#include <math.h>

/* Function to get no of set bits in binary
```

```
   representation of passed binary no.
   Please refer below for details of this
   function :
   https://www.geeksforgeeks.org/count-set-bits-in-an-integer/ */
int countSetBits(int n)
{
    unsigned int count = 0;
    while (n)
    {
      n &= (n-1) ;
      count++;
    }
    return count;
}

// Check if number is odious or not
int checkOdious(int n)
{
    return (countSetBits(n) % 2 == 1);
}

// Driver Code
int main()
{
    int num = 32;
    if (checkOdious(num))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // Java program to check if a number is
// Odious Number or not
import java.io.*;
import java.math.*;

class GFG {

    /* Function to get no of set bits in binary
       representation of passed binary no.
       Please refer below for details of this
       function :
       https://www.geeksforgeeks.org/count-set-bits-in-an-integer/ */
    static int countSetBits(int n)
    {
```

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

    // Check if number is odious or not
    static boolean checkOdious(int n)
    {
        return (countSetBits(n) % 2 == 1);
    }

    // Driver Code
    public static void main(String args[])
    {
        int num = 32;
        if (checkOdious(num))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

/*This code is contributed by Nikita Tiwari.*/
```

Python3

```
 # Python 3 program to check if a number is
# Odious Number or not

# Function to get no of set bits in binary
# representation of passed binary no.
# Please refer below for details of this function :
# https://www.geeksforgeeks.org/count-set-bits-in-an-integer
def countSetBits(n) :
    count = 0

    while (n) :
        n = n & (n-1)
        count = count + 1

    return count


# Check if number is odious or not
```

```
def checkOdious(n) :
    return (countSetBits(n) % 2 == 1)

# Driver Code
num = 32

if (checkOdious(num)) :
    print("Yes")
else :
    print("No")


# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# program to check if a number
// is Odious Number or not
using System;

class GFG {

    /* Function to get no of set bits in
    binary representation of passed binary
    no. Please refer below for details
    of this function :
    https://www.geeksforgeeks.org/count-set-bits-in-an-integer/ */
    static int countSetBits(int n)
    {
        int count = 0;
        while (n != 0)
        {
        n &= (n - 1) ;
        count++;
        }

        return count;
    }

    // Check if number is odious or not
    static bool checkOdious(int n)
    {
        return (countSetBits(n) % 2 == 1);
    }

    // Driver Code
    public static void Main()
    {
```

```
        int num = 32;
        if (checkOdious(num))
            Console.Write("Yes");
        else
            Console.Write("No");
    }
}

/*This code is contributed by vt_m.*/
```

PHP

```
 <?php
// PHP program to check if a number
// is Odious Number or not

// Function to get no of
// set bits in binary
function countSetBits($n)
{
    $count = 0;
    while ($n)
    {
        $n &= ($n - 1) ;
        $count++;
    }
    return $count;
}

// Check if number is odious or not
function checkOdious($n)
{
    return (countSetBits($n) % 2 == 1);
}

// Driver Code
$num = 32;
if (checkOdious($num))
    echo "Yes";
else
    echo "No";

// This code is contributed by mits
?>
```

Output :

```
Yes
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/odious-number/

Chapter 244

Operators in C | Set 2 (Relational and Logical Operators)

Operators in C | Set 2 (Relational and Logical Operators) - GeeksforGeeks

We have discussed introduction to operators in C and Arithmetic Operators. In this article, Relational and Logical Operators are discussed.

Relational Operators:
Relational operators are used for comparison of two values. Let's see them one by one:

- **'=='** operator checks whether the two given operands are equal or not. If so, it returns true. Otherwise it returns false. For example, **5==5** will return true.
- **'!='** operator checks whether the two given operands are equal or not. If not, it returns true. Otherwise it returns false. It is the exact boolean complement of the **'=='** operator. For example, **5!=5** will return false.
- **'>'** operator checks whether the first operand is greater than the second operand. If so, it returns true. Otherwise it returns false. For example, **6>5** will return true.
- **'<'** operator checks whether the first operand is lesser than the second operand. If so, it returns true. Otherwise it returns false. For example, **6<5** will return false.
- **'>='** operator checks whether the first operand is greater than or equal to the second operand. If so, it returns true. Otherwise it returns false. For example, **5>=5** will return true.
- **'<='** operator checks whether the first operand is lesser than or equal to the second operand. If so, it returns true. Otherwise it returns false. For example, **5<=5** will also return true.

```
 // C program to demonstrate working of relational operators
#include <stdio.h>
```

```
int main()
{
    int a=10, b=4;

    // relational operators
    // greater than example
    if (a > b)
        printf("a is greater than b\n");
    else printf("a is less than or equal to b\n");

    // greater than equal to
    if (a >= b)
        printf("a is greater than or equal to b\n");
    else printf("a is lesser than b\n");

    // less than example
    if (a < b)
        printf("a is less than b\n");
    else printf("a is greater than or equal to b\n");

    // lesser than equal to
    if (a <= b)
        printf("a is lesser than or equal to b\n");
    else printf("a is greater than b\n");

    // equal to
    if (a == b)
        printf("a is equal to b\n");
    else printf("a and b are not equal\n");

    // not equal to
    if (a != b)
        printf("a is not equal to b\n");
    else printf("a is equal b\n");

    return 0;
}
```

Output:

```
a is greater than b
a is greater than or equal to b
a is greater than or equal to b
a is greater than b
a and b are not equal
a is not equal to b
```

Logical Operators:
They are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration. They are described below:

- **Logical AND:** The **'&&'** operator returns true when both the conditions in consideration are satisfied. Otherwise it returns false. For example, **a && b** returns true when both a and b are true (i.e. non-zero).
- **Logical OR:** The **'||'** operator returns true when one (or both) of the conditions in consideration is satisfied. Otherwise it returns false. For example, **a || b** returns true if one of a or b is true (i.e. non-zero). Of course, it returns true when both a and b are true.
- **Logical NOT:** The **'!'** operator returns true the condition in consideration is not satisfied. Otherwise it returns false. For example, **!a** returns true if a is false, i.e. when a=0.

```
 // C program to demonstrate working of logical operators
#include <stdio.h>

int main()
{
    int a=10, b=4, c = 10, d = 20;

    // logical operators

    // logical AND example
    if (a>b && c==d)
        printf("a is greater than b AND c is equal to d\n");
    else printf("AND condition not satisfied\n");

    // logical AND example
    if (a>b || c==d)
        printf("a is greater than b OR c is equal to d\n");
    else printf("Neither a is greater than b nor c is equal "
                " to d\n");

    // logical NOT example
    if (!a)
        printf("a is zero\n");
    else printf("a is not zero");

    return 0;
}
```

Output:

```
AND condition not satisfied
```

```
a is greater than b OR c is equal to d
a is not zero
```

Short-Circuiting in Logical Operators:
In case of **logical AND**, the second operand is not evaluated if first operand is false. For example, program 1 below doesn't print "GeeksQuiz" as the first operand of logical AND itself is false.

```
 #include <stdio.h>
#include <stdbool.h>
int main()
{
    int a=10, b=4;
    bool res = ((a == b) && printf("GeeksQuiz"));
    return 0;
}
```

But below program prints "GeeksQuiz" as first operand of logical AND is true.

```
 #include <stdio.h>
#include <stdbool.h>
int main()
{
    int a=10, b=4;
    bool res = ((a != b) && printf("GeeksQuiz"));
    return 0;
}
```

In case of **logical OR**, the second operand is not evaluated if first operand is true. For example, program 1 below doesn't print "GeeksQuiz" as the first operand of logical OR itself is true.

```
 #include <stdio.h>
#include <stdbool.h>
int main()
{
    int a=10, b=4;
    bool res = ((a != b) || printf("GeeksQuiz"));
    return 0;
}
```

But below program prints "GeeksQuiz" as first operand of logical OR is false.

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

```
int main()
{
    int a=10, b=4;
    bool res = ((a == b) || printf("GeeksQuiz"));
    return 0;
}
```

Quiz on Operators in C

This article is contributed by Ayush Jaggi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/operators-in-c-set-2-relational-and-logical-operators/

Chapter 245

Optimization Techniques | Set 1 (Modulus)

Optimization Techniques | Set 1 (Modulus) - GeeksforGeeks

Modulus operator is costly.

The modulus operator (%) in various languages is costly operation. Ultimately every operator/operation must result in processor instructions. Some processors won't have modulus instruction at hardware level, in such case the compilers will insert stubs (predefined functions) to perform modulus. It impacts performance.

There is simple technique to extract remainder when a number is divided by another number (divisor) that is power of 2? A number that is an exact power of 2 will have only one bit set in it's binary representation. Consider the following powers of 2 and their binary representations

2 – 10

4 – 100

8 – 1000

16 – 10000

Note those zeros in red color, they contribute to remainder in division operation. We can get mask for those zeros by decrementing the divisor by 1.

Generalizing the above pattern, a number that can be written in 2^n form will have only one bit set followed by n zeros on the right side of 1. When a number (N) divided by (2^n), the bit positions corresponding to the above mentioned *zeros* will contribute to the remainder of division operation. An example can make it clear,

N = 87 (1010111 – binary form)

N%2 = N & (2-1) = 1010111 & 1 = 1 = 1

N%4 = N & (4-1) = 1010111 & 11 = 11 = 3

N%8 = N & (8-1) = 1010111 & 111 = 111 = 7

N%16 = N & (16-1) = 1010111 & 1111 = 111 = 7

N%32 = N & (32-1) = 1010111 & 11111 = 10111 = 23

Modulus operation over exact powers of 2 is simple and faster bitwise ANDing. This is the reason, programmers usually make buffer length as powers of 2.

Note that the technique will work only for divisors that are powers of 2.

An Example:

Implementation of circular queue (ring buffer) using an array. Omitting one position in the circular buffer implementation can make it easy to distinguish between *full* and *empty* conditions. When the buffer reaches SIZE-1, it needs to wrap back to initial position. The wrap back operation can be simple AND operation if the buffer size is power of 2. If we use any other size, we would need to use modulus operation.

Note:

Per experts comments, premature optimization is an evil. The optimization techniques provided are to fine tune your code after finalizing design strategy, algorithm, data structures and implementation. We recommend to avoid them at the start of code development. Code readability is key for maintenance.

Thanks to Venki for writing the above article. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/optimization-techniques-set-1-modulus/

Chapter 246

Pairs of complete strings in two sets of strings

Pairs of complete strings in two sets of strings - GeeksforGeeks

Two strings are said to be complete if on concatenation, they contain all the 26 English alphabets. For example, “abcdefghi” and “jklmnopqrstuvwxyz” are complete as they together have all characters from ‘a’ to ‘z’.
We are given two sets of sizes n and m respectively and we need to find the number of pairs that are complete on concatenating each string from set 1 to each string from set 2.

```
Input : set1[] = {"abcdefgh", "geeksforgeeks",
                  "lmnopqrst", "abc"}
        set2[] = {"ijklmnopqrstuvwxyz",
                  "abcdefghijklmnopqrstuvwxyz",
                  "defghijklmnopqrstuvwxyz"}
Output : 7
The total complete pairs that are forming are:
"abcdefghijklmnopqrstuvwxyz"
"abcdefghabcdefghijklmnopqrstuvwxyz"
"abcdefghdefghijklmnopqrstuvwxyz"
"geeksforgeeksabcdefghijklmnopqrstuvwxyz"
"lmnopqrstabcdefghijklmnopqrstuvwxyz"
"abcabcdefghijklmnopqrstuvwxyz"
"abcdefghijklmnopqrstuvwxyz"
```

Method 1 (Naive method)

A simple solution is to consider all pairs of strings, concatenate them and then check if the concatenated string has all the characters from ‘a’ to ‘z’ by using a frequency array.

```
// C++ implementation for find pairs of complete
```

```
// strings.
#include<iostream>
using namespace std;

// Returns count of complete pairs from set[0..n-1]
// and set2[0..m-1]
int countCompletePairs(string set1[], string set2[],
                       int n, int m)
{
    int result = 0;

    // Consider all pairs of both strings
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<m; j++)
        {
            // Create a concatenation of current pair
            string concat = set1[i] + set2[j];

            // Compute frequencies of all characters
            // in the concatenated string.
            int frequency[26] = {0};
            for (int k=0; k<concat.length(); k++)
                frequency[concat[k] - 'a']++;

            // If frequency of any character is not
            // greater than 0, then this pair is not
            // complete.
            int i;
            for (i=0; i<26; i++)
                if (frequency[i] < 1)
                    break;
            if (i == 26)
                result++;
        }
    }

    return result;
}

// Driver code
int main()
{
    string set1[] = {"abcdefgh", "geeksforgeeks",
                     "lmnopqrst", "abc"};
    string set2[] = {"ijklmnopqrstuvwxyz",
                     "abcdefghijklmnopqrstuvwxyz",
                     "defghijklmnopqrstuvwxyz"};
```

```
    int n = sizeof(set1)/sizeof(set1[0]);
    int m = sizeof(set2)/sizeof(set2[0]);

    cout << countCompletePairs(set1, set2, n, m);

    return 0;
}
```

Output:

```
7
```

Method 2 (Optimized method using Bit Manipulation)

In this method, we compress frequency array into an integer. We assign each bit of that integer with a character and we set it to 1 when the character is found. We perform this for all the strings in both the sets. Finally we just compare the two integers in the sets and if on combining all the bits are set, they form a complete string pair.

```
 // C++ program to find count of complete pairs
#include<iostream>
using namespace std;

// Returns count of complete pairs from set[0..n-1]
// and set2[0..m-1]
int countCompletePairs(string set1[], string set2[],
                       int n, int m)
{
    int result = 0;

    // con_s1[i] is going to store an integer whose
    // set bits represent presence/absence of characters
    // in string set1[i].
    // Similarly con_s2[i] is going to store an integer
    // whose set bits represent presence/absence of
    // characters in string set2[i]
    int con_s1[n], con_s2[m];

    // Process all strings in set1[]
    for (int i=0; i<n; i++)
    {
        // initializing all bits to 0
        con_s1[i] = 0;
        for (int j=0; j<set1[i].length(); j++)
        {
```

```
            // Setting the ascii code of char s[i][j]
            // to 1 in the compressed integer.
            con_s1[i] = con_s1[i] | (1<<(set1[i][j]-'a'));
        }
    }

    // Process all strings in set2[]
    for (int i=0; i<m; i++)
    {
        // initializing all bits to 0
        con_s2[i] = 0;
        for (int j=0; j<set2[i].length(); j++)
        {
            // setting the ascii code of char s[i][j]
            // to 1 in the compressed integer.
            con_s2[i] = con_s2[i]|(1<<(set2[i][j]-'a'));
        }
    }

    // assigning a variable whose all 26 (0..25)
    // bits are set to 1
    long long complete = (1<<26) - 1;

    // Now consider every pair of integer in con_s1[]
    // and con_s2[] and check if the pair is complete.
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<m; j++)
        {
            // if all bits are set, the strings are
            // complete!
            if ((con_s1[i] | con_s2[j]) == complete)
                result++;
        }
    }

    return result;
}

// Driver code
int main()
{
    string set1[] = {"abcdefgh", "geeksforgeeks",
                     "lmnopqrst", "abc"
                    };
    string set2[] = {"ijklmnopqrstuvwxyz",
                     "abcdefghijklmnopqrstuvwxyz",
                     "defghijklmnopqrstuvwxyz"
```

```
                    };
    int n = sizeof(set1)/sizeof(set1[0]);
    int m = sizeof(set2)/sizeof(set2[0]);

    cout << countCompletePairs(set1, set2, n, m);

    return 0;
}
```

Output:

```
7
```

Source

https://www.geeksforgeeks.org/pairs-of-complete-strings-in-two-sets-of-strings/

Chapter 247

Pairs whose concatenation contain all digits

Pairs whose concatenation contain all digits - GeeksforGeeks

Given an array of **n** numbers. The task is to find the number of pairs that can be taken from the given which on concatenation will contain all the digits from 0 to 9.

Examples:

> Input : num[][] = { “129300455”, “5559948277”, “012334556”, “56789”, “123456879” }
> Output : 5
> {“129300455”, “56789”}, { “129300455”, “123456879”}, {“5559948277”, “012334556”},
> {“012334556”, “56789”}, {“012334556”, “123456879”} are the pair which contain all the digits from 0 to 9 on concatenation.

Note: Number of digit in each of the number can be 10^6.

The idea is to represent each number as the mask of 10 bits such that if it contains digit **i** at least once then $\mathbf{i^{th}}$ bit will be set in the mask.
For example,
let n = 4556120 then 0^{th}, 1^{st}, 2^{nd}, 4^{th}, 5^{th}, 6^{th} bits will be set in mask.
Thus, mask = $(0001110111)_2 = (119)_{10}$
Now, for every mask **m** from 0 to 2^10 – 1, we will store the count of the number of numbers having the mask of their number equals to **m**.
So, we will make an array, say cnt[], where cnt[i] stores the count of the number of numbers whose mask is equal to i. Pseudocode for this:

```
for (i = 0; i < (1 << 10); i++)
  cnt[i] = 0;
```

```
for (i = 1; i <= n; i++)
{
  string x = p[i];
  int mask  = 0;
  for (j = 0; j < x.size(); j++)
    mask |= (1 << (x[j] - '0';);


  cnt[mask]++;
}
```

A pair of numbers will have all the digit from 0 to 9 if every bits from 0 to 9 is set in the bitwise OR of maskof both the number, i.e if it's equal to $(1111111111)_{2</sub)} = (1023)_{10}$

Now, we will iterate over all pairs of masks whose biwise OR is equal to 1023 and add the number of ways.

Below is C++ implementation of this approach:

```
 // CPP Program to find number of pairs whose
// concatenation contains all digits from 0 to 9.
#include <bits/stdc++.h>
using namespace std;
#define N 20

// Function to return number of pairs whose
// concatenation contain all digits from 0 to 9
int countPair(char str[N][N], int n)
{
    int cnt[1 << 10] = { 0 };

    // making the mask for each of the number.
    for (int i = 0; i < n; i++) {

        int mask = 0;
        for (int j = 0; str[i][j] != '\0'; ++j)
            mask |= (1 << (str[i][j] - '0'));
        cnt[mask]++;
    }

    // for each of the possible pair which can
    // make OR value equal to 1023
    int ans = 0;
    for (int m1 = 0; m1 <= 1023; m1++)
        for (int m2 = 0; m2 <= 1023; m2++)
            if ((m1 | m2) == 1023) {

                // finding the count of pair
                // from the given numbers.
```

```
                ans += ((m1 == m2) ?
                       (cnt[m1] * (cnt[m1] - 1)) :
                       (cnt[m1] * cnt[m2]));
            }

    return ans / 2;
}

// Driven Program
int main()
{
    int n = 5;
    char str[][N] = { "129300455", "5559948277",
                "012334556", "56789", "123456879" };
    cout << countPair(str, n) << endl;
    return 0;
}
```

Output:

```
5
```

Complexity : O(n + 2^10 * 2^10)

Source

https://www.geeksforgeeks.org/pairs-whose-concatenation-contain-digits/

Chapter 248

Pernicious number

Pernicious number - GeeksforGeeks

A **pernicious number** is a positive integer which has *prime number of ones in its binary representation*. The first pernicious number is 3 since 3 = (11)(in binary representation) and 1 + 1 = 2, which is a prime.

Properties of Pernicious Numbers :

1. There isn't any pernicious number which is also *power of 2* because powers of two in binary form are represented as a one followed by zeros. So, 1 is not considered as a prime number.
2. Every number of the form $2^n + 1$ with n > 0 is a pernicious number as the number of ones in binary form is 2 which is prime.
3. A number of the form $2^p - 1$ with prime p is a pernicious number known as a Mersenne number .

The idea to print first n Pernicious numbers is simple.

Do following for every number from 1 to n.
1) Count set bits in current number
2) Print current number if count of set bits is prime. We use simple primality check for this purpose.

Here is the program to print first 25 pernicious number.

Below is the implementation of the above approach.

C++

```
 // CPP program to print first n pernicious numbers
#include <bits/stdc++.h>
using namespace std;

// function to check prime number
```

```
bool isPrime(int x)
{
    if (x < 2)
        return false;
    for (int i = 2; i < x; i++) {
        if (x % i == 0)
            return false;
    }
    return true;
}

// Prints first n Pernicious numbers
void printPernicious(int n)
{
    for (int i=1,count=0; count<n; i++) {

        // "__builtin_popcount(i)" count no
        // of ones in binary representation
        if (isPrime(__builtin_popcount(i))) {
            cout << i << " ";

            count++;
        }
    }
}

int main()
{
   int n = 25;
   printPernicious(n);
   return 0;
}
```

Java

```
 // Java program to print first
// n pernicious numbers
import java.util.*;

class GFG {
    // function to count no of
    // ones in binary representation
    static int countSetBits(int n)
    {
        int count = 0;

        while (n > 0)
        {
```

```
            n &= (n - 1) ;
            count++;
        }
        return count;
    }

    // function to check prime number
    static boolean isPrime(int x)
    {
        if (x < 2)
            return false;
        for (int i = 2; i < x; i++) {
            if (x % i == 0)
                return false;
        }
        return true;
    }

    // Prints first n Pernicious numbers
    static void printPernicious(int n)
    {
        for (int i=1,count=0; count<n; i++) {

            if (isPrime(countSetBits(i))) {
                System.out.print( i + " ");

                count++;
            }
        }
    }

    // Driver Code
    public static void main (String[] args) {
        int n = 25;
        printPernicious(n);
    }
}

// This code is contributed by Ansu Kumari
```

Python3

```
 # Python program to print
# first n pernicious numbers

# function to check
# prime number
def isPrime(x):
```

```
    if x < 2:
        return False

    for i in range(2, x):
        if not x % i:
            return False

    return True

# Prints first n Pernicious
# numbers
def printPernicious(n):

    i, count = 1, 0

    while count < n:

        # "bin(i).count('1')" count
        # no of ones in binary
        # representation
        if (isPrime(bin(i).count('1'))):
            print(i, end=' ')
            count += 1

        i += 1

# Driver Code
n = 25
printPernicious(n)

# This code is contributed by Ansu Kumari
```

C#

```
 // C#program to print first
// n pernicious numbers
using System;

class GFG
{
    // function to count no of
    // ones in binary representation
    static int countSetBits(int n)
    {
        int count = 0;

        while (n > 0)
```

```
        {
            n &= (n - 1) ;
            count++;
        }
        return count;
    }

    // function to check prime number
    static bool isPrime(int x)
    {
        if (x < 2)
            return false;
        for (int i = 2; i < x; i++) {
            if (x % i == 0)
                return false;
        }
        return true;
    }

    // Prints first n Pernicious numbers
    static void printPernicious(int n)
    {
        for (int i=1,count=0; count<n; i++) {

            if (isPrime(countSetBits(i))) {
                Console.Write( i + " ");

                count++;
            }
        }
    }

    // Driver Code
    public static void Main ()
    {
        int n = 25;
        printPernicious(n);
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP program to print first
// n pernicious numbers
```

```
// function to check prime number
function isPrime($x)
{
    if ($x < 2)
        return false;
    for ($i = 2; $i < $x; $i++)
    {
        if ($x % $i == 0)
            return false;
    }
    return true;
}
//this function count no of
// ones in binary representation
function getBitCount($value)
{

    $count = 0;
    while($value)
    {
        $count += ($value & 1);
        $value = $value >> 1;
    }

    return $count;
}

// Prints first n Pernicious numbers
function printPernicious($n)
{
    for ($i = 1, $count = 0;
                $count < $n; $i++)
    {

        //count no of ones in
        // binary representation
        if (isPrime(getBitCount($i)))
        {
            echo $i." ";

            $count++;
        }
    }
}

// Driver code
$n = 25;
printPernicious($n);
```

```
// This code is contributed by mits
?>
```

Output :

```
3 5 6 7 9 10 11 12 13 14 17 18 19 20 21 22 24 25 26 28 31 33 34 35 36
```

References :
Wiki

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/pernicious-number/

Chapter 249

Position of rightmost bit with first carry in sum of two binary

Position of rightmost bit with first carry in sum of two binary - GeeksforGeeks

Given two non-negative integers **a** and **b**. The problem is to find the position of the rightmost bit where a carry is generated in the binary addition of **a** and **b**.

Examples:

```
Input : a = 10, b = 2
Output : 2
(10)10 = (1010)2
(2)10 = (10)2.
  1010
+   10
As highlighted, 1st carry bit from the right
will be generated at position '2'.

Input : a = 10, b = 5
Output : 0
'0' as no carry bit will be generated.
```

Approach: Following are the steps:

1. Calculate **num** = a & b.
2. Find the position of rightmost set bit in **num**.

CPP

```
// C++ implementation to find the position of
```

```
// rightmost bit where a carry is generated first
#include <bits/stdc++.h>
using namespace std;

typedef unsigned long long int ull;

// function to find the position of
// rightmost set bit in 'n'
unsigned int posOfRightmostSetBit(ull n)
{
    return log2(n & -n) + 1;
}

// function to find the position of rightmost
// bit where a carry is generated first
unsigned int posOfCarryBit(ull a, ull b)
{
    return posOfRightmostSetBit(a & b);
}

// Driver program to test above
int main()
{
    ull a = 10, b = 2;
    cout << posOfCarryBit(a, b);
    return 0;
}
```

Java

```
 // Java implementation to find the position of
// rightmost bit where a carry is generated first
class GFG {

    // function to find the position of
    // rightmost set bit in 'n'
    static int posOfRightmostSetBit(int n)
    {
        return (int)(Math.log(n & -n) / Math.log(2)) + 1;
    }

    // function to find the position of rightmost
    // bit where a carry is generated first
    static int posOfCarryBit(int a, int b)
    {
        return posOfRightmostSetBit(a & b);
    }
```

```
    // Driver code
    public static void main(String[] args)
    {
        int a = 10, b = 2;

        System.out.print(posOfCarryBit(a, b));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 implementation to find the position of
# rightmost bit where a carry is generated first

import math

# function to find the position of
# rightmost set bit in 'n'
def posOfRightmostSetBit( n ):
    return int(math.log2(n & -n) + 1)

# function to find the position of rightmost
# bit where a carry is generated first
def posOfCarryBit( a , b ):
    return posOfRightmostSetBit(a & b)

# Driver program to test above
a = 10
b = 2
print(posOfCarryBit(a, b))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# implementation to find the position of
// rightmost bit where a carry is generated first
using System;

class GFG {

    // function to find the position of
    // rightmost set bit in 'n'
    static int posOfRightmostSetBit(int n)
    {
```

```
        return (int)(Math.Log(n & -n) / Math.Log(2)) + 1;
    }

    // function to find the position of rightmost
    // bit where a carry is generated first
    static int posOfCarryBit(int a, int b)
    {
        return posOfRightmostSetBit(a & b);
    }

    // Driver code
    public static void Main()
    {
        int a = 10, b = 2;

        Console.Write(posOfCarryBit(a, b));
    }
}

// This code is contributed by Sam007.
```

Output:

```
2
```

Source

https://www.geeksforgeeks.org/position-rightmost-bit-first-carry-sum-two-binary/

Chapter 250

Position of rightmost common bit in two numbers

Position of rightmost common bit in two numbers - GeeksforGeeks

Given two non-negative numbers **m** and **n**. Find the position of rightmost same bit in the binary representation of the numbers.

Examples:

```
Input : m = 10, n = 9
Output : 3
(10)10 = (1010)2
(9)10 = (1001)2
It can be seen that the 3rd bit
from the right is same.

Input : m = 16, n = 7
Output : 4
(16)10 = (10000)2
(7)10 = (111)2, can also be written as
     = (00111)2
It can be seen that the 4th bit
from the right is same.
```

Approach: Get the bitwise xor of **m** and **n**. Let it be **xor_value** = m ^ n. Now, get the position of rightmost unset bit in **xor_value**.

Explanation: The bitwise xor operation produces a number which has unset bits only at the positions where the bits of **m** and **n** are same. Thus, the position of rightmost unset bit in **xor_value** gives the position of rightmost same bit.

C++

```
 // C++ implementation to find the position
// of rightmost same bit
#include <bits/stdc++.h>

using namespace std;

// Function to find the position of
// rightmost set bit in 'n'
int getRightMostSetBit(unsigned int n)
{
    return log2(n & -n) + 1;
}

// Function to find the position of
// rightmost same bit in the
// binary representations of 'm' and 'n'
int posOfRightMostSameBit(unsigned int m,
                          unsigned int n)
{
    // position of rightmost same bit
    return getRightMostSetBit(~(m ^ n));
}

// Driver program to test above
int main()
{
    int m = 16, n = 7;
    cout << "Position = "
         << posOfRightMostSameBit(m, n);
    return 0;
}
```

Java

```
 // Java implementation to find the position
// of rightmost same bit
class GFG {

    // Function to find the position of
    // rightmost set bit in 'n'
    static int getRightMostSetBit(int n)
    {
        return (int)((Math.log(n & -n))/(Math.log(2)))
                                                + 1;
    }

    // Function to find the position of
    // rightmost same bit in the
```

```
    // binary representations of 'm' and 'n'
    static int posOfRightMostSameBit(int m,int n)
    {

        // position of rightmost same bit
        return getRightMostSetBit(~(m ^ n));
    }

    //Driver code
    public static void main (String[] args)
    {
        int m = 16, n = 7;

        System.out.print("Position = "
            + posOfRightMostSameBit(m, n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 implementation to find the
# position of rightmost same bit
import math

# Function to find the position
# of rightmost set bit in 'n'
def getRightMostSetBit(n):

    return int(math.log2(n & -n)) + 1

# Function to find the position of
# rightmost same bit in the binary
# representations of 'm' and 'n'
def posOfRightMostSameBit(m, n):

    # position of rightmost same bit
    return getRightMostSetBit(~(m ^ n))

# Driver Code
m, n = 16, 7
print("Position = ", posOfRightMostSameBit(m, n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# implementation to find the position
// of rightmost same bit
using System;

class GFG
{
    // Function to find the position of
    // rightmost set bit in 'n'
    static int getRightMostSetBit(int n)
    {
        return (int)((Math.Log(n & -n)) / (Math.Log(2))) + 1;
    }

    // Function to find the position of
    // rightmost same bit in the
    // binary representations of 'm' and 'n'
    static int posOfRightMostSameBit(int m,int n)
    {
        // position of rightmost same bit
        return getRightMostSetBit(~(m ^ n));
    }

    //Driver code
    public static void Main ()
    {
        int m = 16, n = 7;
        Console.Write("Position = "
              + posOfRightMostSameBit(m, n));
    }
}
//This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP implementation to
// find the position
// of rightmost same bit

// Function to find the position of
// rightmost set bit in 'n'
function getRightMostSetBit($n)
{
    return log($n & -$n) + 1;
}

// Function to find the position of
// rightmost same bit in the
```

```
// binary representations of 'm' and 'n'
function posOfRightMostSameBit($m,
                               $n)
{

    // position of rightmost same bit
    return getRightMostSetBit(~($m ^ $n));
}

    // Driver Code
    $m = 16; $n = 7;
    echo "Position = "
        , ceil(posOfRightMostSameBit($m, $n));

// This code is contributed by anuj_67.
?>
```

Output:

```
Position = 4
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/position-rightmost-common-bit-two-numbers/

Chapter 251

Position of rightmost different bit

Position of rightmost different bit - GeeksforGeeks

Given two numbers **m** and **n**. Find the position of rightmost different bit in binary representation of numbers. It is guaranteed that such a bit exists.

Examples :

```
Input : m = 11, n = 9
Output : 2
(11)10 = (1011)2
(9)10 = (1001)2
It can be seen that 2nd bit from
the right is different

Input : m = 52, n = 4
Output : 5
(52)10 = (110100)2
(4)10 = (100)2, can also be written as
     = (000100)2
It can be seen that 5th bit from
the right is different
```

Approach: Get the bitwise xor of **m** and **n**. Let it be **xor_value** = m ^ n. Now, find the position of rightmost set bit in **xor_value**.

Explanation: The bitwise xor operation produces a number which has set bits only at the positions where the bits of **m** and **n** differ. Thus, the position of rightmost set bit in **xor_value** gives the position of rightmost different bit.

C++

```
 // C++ implementation to find the position
// of rightmost different bit
#include <bits/stdc++.h>
using namespace std;

// Function to find the position of
// rightmost set bit in 'n'
int getRightMostSetBit(int n)
{
    return log2(n & -n) + 1;
}

// Function to find the position of
// rightmost different bit in the
// binary representations of 'm' and 'n'
int posOfRightMostDiffBit(int m, int n)
{
    // position of rightmost different
    //  bit
    return getRightMostSetBit(m ^ n);
}

// Driver program
int main()
{
    int m = 52, n = 4;
    cout << posOfRightMostDiffBit(m, n);
    return 0;
}
```

Java

```
 // Java implementation to find the position
// of rightmost different bit

class GFG {

    // Function to find the position of
    // rightmost set bit in 'n'
    static int getRightMostSetBit(int n)
    {
        return (int)((Math.log10(n & -n)) / Math.log10(2)) + 1;
    }

    // Function to find the position of
    // rightmost different bit in the
    // binary representations of 'm' and 'n'
    static int posOfRightMostDiffBit(int m, int n)
```

```
    {
        // position of rightmost different bit
        return getRightMostSetBit(m ^ n);
    }

    // Driver code
    public static void main(String arg[])
    {
        int m = 52, n = 4;
        System.out.print("Position = " +
            posOfRightMostDiffBit(m, n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python implementation
# to find the position
# of rightmost different bit

import math

# Function to find the position of
# rightmost set bit in 'n'
def getRightMostSetBit(n):

    return math.log2(n & -n) + 1


# Function to find the position of
# rightmost different bit in the
# binary representations of 'm' and 'n'
def posOfRightMostDiffBit(m, n):

    # position of rightmost different
    # bit
    return getRightMostSetBit(m ^ n)

# Driver code

m = 52
n = 4
print("position = ", int(posOfRightMostDiffBit(m, n)))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C#implementation to find the position
// of rightmost different bit
using System;

class GFG {

    // Function to find the position of
    // rightmost set bit in 'n'
    static int getRightMostSetBit(int n)
    {
        return (int)((Math.Log10(n & -n))
                       / Math.Log10(2)) + 1;
    }

    // Function to find the position of
    // rightmost different bit in the
    // binary representations of 'm' and 'n'
    static int posOfRightMostDiffBit(int m, int n)
    {
        // position of rightmost different bit
        return getRightMostSetBit(m ^ n);
    }

    // Driver code
    public static void Main()
    {
        int m = 52, n = 4;
        Console.Write("Position = " +
            posOfRightMostDiffBit(m, n));
    }
}

// This code is contributed by Smitha.
```

PHP

```
 <?php
// PHP implementation to
// find the position of
// rightmost different bit

// Function to find the position
// of rightmost set bit in 'n'
function getRightMostSetBit($n)
{
```

```
    return log($n & -$n, (2)) + 1;
}

// Function to find the position of
// rightmost different bit in the
// binary representations of 'm'
// and 'n'
function posOfRightMostDiffBit($m, $n)
{

    // position of rightmost
    // different bit
    return getRightMostSetBit($m ^ $n);
}

    // Driver Code
    $m = 52;
    $n = 4;
    echo posOfRightMostDiffBit($m, $n);

// This code is contributed by Ajit
?>
```

Output:

```
5
```

Using ffs() function

C++

```
 // C++ implementation to find the
// position of rightmost different
// bit in two number.
#include <bits/stdc++.h>
using namespace std;

// function to find rightmost different
//  bit in two numbers.
int posOfRightMostDiffBit(int m, int n)
{
    return ffs(m ^ n);
}

// Driver code
int main()
{
```

```
    int m = 52, n = 4;
    cout <<"Position = " <<
         posOfRightMostDiffBit(m, n);
    return 0;
}
```

PHP

```
 <?php
// PHP implementation to find the
// position of rightmost different
// bit in two number.

// function to find rightmost
// different bit in two numbers.
function posOfRightMostDiffBit($m, $n)
{
    $t = floor(log($m ^ $n, 2));
    return $t;
}

// Driver code
$m = 52;
$n = 4;
echo "Position = " ,
    posOfRightMostDiffBit($m, $n);

// This code is contributed by ajit
?>
```

Output :

```
Position = 5
```

Improved By : Mannuyou, jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/position-rightmost-different-bit/

Chapter 252

Position of rightmost set bit

Position of rightmost set bit - GeeksforGeeks

Write a one line C function to return position of first 1 from right to left, in binary representation of an Integer.

```
I/P    18,   Binary Representation 010010
O/P   2
I/P    19,   Binary Representation 010011
O/P   1
```

```
Algorithm: (Example 18(010010))
Let I/P be 12 (1100)

1. Take two's complement of the given no as all bits are reverted
except the first '1' from right to left (10111)

2  Do an bit-wise & with original no, this will return no with the
required one only (00010)

3  Take the log2 of the no, you will get position -1 (1)

4  Add 1 (2)
```

Program:

C

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

```
unsigned int getFirstSetBitPos(int n)
{
    return log2(n & -n) + 1;
}

int main()
{
    int n = 12;
    printf("%u", getFirstSetBitPos(n));
    getchar();
    return 0;
}
```

Java

```
 // Java Code for Position of rightmost set bit
class GFG {

    public static int getFirstSetBitPos(int n)
    {
        return (int)((Math.log10(n & -n)) / Math.log10(2)) + 1;
    }

    // Drive code
    public static void main(String[] args)
    {
        int n = 12;
        System.out.println(getFirstSetBitPos(n));
    }
}
// This code is contributed by Arnav Kr. Mandal
```

Python3

```
 # Python Code for Position
# of rightmost set bit

import math

def getFirstSetBitPos(n):

     return math.log2(n&-n)+1

# driver code

n = 12
```

```
print(int(getFirstSetBitPos(n)))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# Code for Position of rightmost set bit
using System;

class GFG {
    public static int getFirstSetBitPos(int n)
    {
        return (int)((Math.Log10(n & -n)) / Math.Log10(2)) + 1;
    }

    // Driver method
    public static void Main()
    {
        int n = 12;
        Console.WriteLine(getFirstSetBitPos(n));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP Code for Position of
// rightmost set bit

function getFirstSetBitPos($n)
{
    return ceil(log(($n& -
                     $n) + 1, 2));
}

// Driver Code
$n = 12;
echo getFirstSetBitPos($n);

// This code is contributed by m_kit
?>
```

Output:

```
3
```

Using ffs() function: ffs() function returns the index of first least significant set bit. The indexing starts in ffs() function from 1.
For example:
n = 12 = 1100
In above example, ffs(n) returns the rightmost set bit index which is 3.

C++

```
 // C++ program to find the
// position of first rightmost
// set bit in a given number.
#include <bits/stdc++.h>
using namespace std;

// Function to find rightmost
// set bit in given number.
int getFirstSetBitPos(int n)
{
    return ffs(n);
}

// Driver function
int main()
{
    int n = 12;
    cout << getFirstSetBitPos(n) << endl;
    return 0;
}
```

Using XOR and & operator :
Initialize m as 1 as check its XOR with the bits starting from the rightmost bit. Left shift m by one till we find the first set bit, as the first set bit gives a number when we perform a & operation with m.

C++

```
 // C++ program to find the first
// rightmost set bit using XOR operator
#include <bits/stdc++.h>
using namespace std;

// function to find the rightmost set bit
int PositionRightmostSetbit(int n)
{
    // Position variable initialize with 1
    // m variable is used to check the set bit
```

```
    int position = 1;
    int m = 1;

    while (!(n & m)) {

        // left shift
        m = m << 1;
        position++;
    }
    return position;
}
// Driver Code
int main()
{
    int n = 16;
    // function call
    cout << PositionRightmostSetbit(n);
    return 0;
}
```

Java

```
 // Java program to find the
// first rightmost set bit
// using XOR operator

class GFG {

    // function to find
    // the rightmost set bit
    static int PositionRightmostSetbit(int n)
    {
        // Position variable initialize
        // with 1 m variable is used to
        // check the set bit
        int position = 1;
        int m = 1;

        while ((n & m) == 0) {

            // left shift
            m = m << 1;
            position++;
        }
        return position;
    }

    // Driver Code
```

```
    public static void main(String[] args)
    {
        int n = 16;

        // function call
        System.out.println(PositionRightmostSetbit(n));
    }
}

// This code is contributed
// by Smitha
```

Python3

```
 # Python3 program to find
# the first rightmost set
# bit using XOR operator

# function to find the
# rightmost set bit
def PositionRightmostSetbit(n):

    # Position variable initialize
    # with 1 m variable is used
    # to check the set bit
    position = 1
    m = 1

    while (not(n & m)) :

        # left shift
        m = m << 1
        position += 1

    return position

# Driver Code
n = 16

# function call
print(PositionRightmostSetbit(n))

# This code is contributed
# by Smitha
```

C#

```
 // C# program to find the
```

```
// first rightmost set bit
// using XOR operator
using System;

class GFG {

    // function to find
    // the rightmost set bit
    static int PositionRightmostSetbit(int n)
    {
        // Position variable initialize
        // with 1 m variable is used to
        // check the set bit
        int position = 1;
        int m = 1;

        while ((n & m) == 0) {

            // left shift
            m = m << 1;
            position++;
        }
        return position;
    }

    // Driver Code
    static public void Main()
    {
        int n = 16;

        // function call
        Console.WriteLine(
            PositionRightmostSetbit(n));
    }
}

// This code is contributed
// by @ajit
```

PHP

```
 <?php
// PHP program to find the
// first rightmost set bit
// using XOR operator

// function to find the
// rightmost set bit
```

```
function PositionRightmostSetbit($n)
{
    // Position variable initialize
    // with 1 m variable is used to
    // check the set bit
    $position = 1;
    $m = 1;

    while (!($n & $m))
    {

        // left shift
        $m = $m << 1;
        $position++;
    }
    return $position;
}

// Driver Code
$n = 16;

// function call
echo PositionRightmostSetbit($n);

// This code is contributed by ajit
?>
```

Output:

```
5
```

This approach has been contributed by mubashshir ahmad

Using Left Shift (<<) : Initialize pos with 1, iterate up to INT_SIZE(Here 32) and check whether bit is set or not, if bit is set then break the loop, else increment the pos.

```
 #include <iostream>
using namespace std;
#define INT_SIZE 32
int Right_most_setbit(int num)
{
    int pos = 1;
    // counting the position of first set bit
    for (int i = 0; i < INT_SIZE; i++) {
        if (!(num & (1 << i)))
            pos++;
        else
```

```
            break;
    }
    return pos;
}
int main()
{
    int num = 18;
    int pos = Right_most_setbit(num);
    cout << pos << endl;
    return 0;
}
// This approach has been contributed by @vivek kumar9
```

Output :

```
2
```

Improved By : Mannuyou, Smitha Dinesh Semwal, jit_t, vivek kumar 9

Source

https://www.geeksforgeeks.org/position-of-rightmost-set-bit/

Chapter 253

Powers of 2 to required sum

Powers of 2 to required sum - GeeksforGeeks

Given an integer N, task is to find the numbers which when raised to the power of 2 and added finally, gives the integer N.

Example :

```
Input : 71307
Output : 0, 1, 3, 7, 9, 10, 12, 16
Explanation :
71307 = 2^0 + 2^1 + 2^3 + 2^7 +
        2^9 + 2^10 + 2^12 + 2^16

Input : 1213
Output : 0, 2, 3, 4, 5, 7, 10
Explanation :
1213 = 2^0 + 2^2 + 2^3 + 2^4 +
       2^5 + 2^7 + 2^10
```

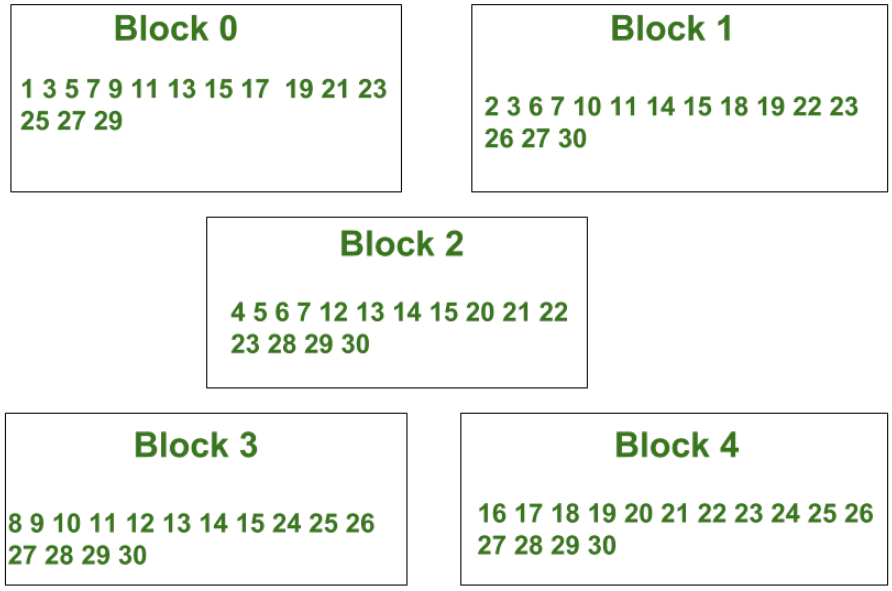
Approach :
Every number can be described in powers of 2.
Example : 29 = 2^0 + 2^2 + 2^3 + 2^4.

2^0 (exponent of 2 is '0') 0
2^2 (exponent of 2 is '2') 1
2^3 (exponent of 2 is '3') 3
2^4 (exponent of 2 is '4') 4

Convert each number into its binary equivalent by pushing remainder of given number, when divided by 2 till it is greater than 0, to vector. Now, Iterate through its binary equivalent and whenever there is set bit, just print the i-th value(iteration number).

Application :
Hamming Code : Hamming Code is an error correcting code which can detect and correct

one bit error. This pattern is also used in Hamming code error detection where parity bits store the XOR of numbers on the basis of LSB(Least Significant bit), where numbers are assigned in blocks and you need to find the blocks where the sum of power of 2 resulting to given number exists. Below is the image to show the blocks with given numbers.



Below is the implementation of above approach :

C++

```
 // CPP program to find the
// blocks for given number.
#include <bits/stdc++.h>
using namespace std;

int block(long int x)
{
    vector<long int> v;

    // Converting the decimal number
    // into its binary equivalent.
    cout << "Blocks for " << x << " : ";
    while (x > 0)
    {
```

```
        v.push_back(x % 2);
        x = x / 2;
    }

    // Displaying the output when
    // the bit is '1' in binary
    // equivalent of number.
    for (int i = 0; i < v.size(); i++)
    {
        if (v[i] == 1)
        {
            cout << i;
            if (i != v.size() - 1)
                cout << ", ";
        }
    }
    cout << endl;
}

// Driver Function
int main()
{
    block(71307);
    block(1213);
    block(29);
    block(100);
    return 0;
}
```

Java

```
 // Java program to find the
// blocks for given number.
import java.util.*;

class GFG {

static void block(long x)
{
    ArrayList<Integer> v = new ArrayList<Integer>();

    // Convert decimal number to
    // its binary equivalent
    System.out.print("Blocks for "+x+" : ");
    while (x > 0)
    {
        v.add((int)x % 2);
        x = x / 2;
```

```
    }

    // Displaying the output when
    // the bit is '1' in binary
    // equivalent of number.
    for (int i = 0; i < v.size(); i++)
    {
        if (v.get(i) == 1)
        {
        System.out.print(i);
            if (i != v.size() - 1)
            System.out.print( ", ");
        }
    }
System.out.println();
}

// Driver Code
public static void main(String args[])
{
    block(71307);
    block(1213);
    block(29);
    block(100);
}
}

// This code is contributed by Arnab Kundu.
```

Python3

```
 # Python3 program to find the
# blocks for given number.
def block(x):

    v = []

    # Converting the decimal number
    # into its binary equivalent.
    print ("Blocks for %d : " %x, end="")
    while (x > 0):
        v.append(int(x % 2))
        x = int(x / 2)

    # Displaying the output when
    # the bit is '1' in binary
    # equivalent of number.
    for i in range(0, len(v)):
```

```
        if (v[i] == 1):
            print (i, end = "")
            if (i != len(v) - 1):
                print (", ", end = "")
    print ("\n")

block(71307)
block(1213)
block(29)
block(100)

# This code is contributed by Manish
# Shaw (manishshaw1)
```

C#

```
 // C# program to find the
// blocks for given number.
using System;
using System.Collections.Generic;

class GFG {

    static void block(long x)
    {
        List<int> v = new List<int>();

        // Convert decimal number to
        // its binary equivalent
        Console.Write("Blocks for " + x + " : ");

        while (x > 0)
        {
            v.Add((int)x % 2);
            x = x / 2;
        }

        // Displaying the output when
        // the bit is '1' in binary
        // equivalent of number.
        for (int i = 0; i < v.Count; i++)
        {
            if (v[i] == 1)
            {
                Console.Write(i);

                if (i != v.Count - 1)
                    Console.Write(", ");
```

```
            }
        }

        Console.WriteLine();
    }

    // Driver Code here
    public static void Main()
    {
        block(71307);
        block(1213);
        block(29);
        block(100);
    }
}

// This code is contributed by Ajit.
```

PHP

```
 <?php
// PHP program to find the
// blocks for given number.

function block($x)
{
    $v = array();

    // Convert decimal number to
    // its binary equivalent
    echo 'Blocks for ' .$x.' : ';

    while ($x > 0)
    {
        array_push($v,intval($x % 2));
        $x = intval($x / 2);
    }

    // Displaying the output when
    // the bit is '1' in binary
    // equivalent of number.
    for ($i = 0; $i < sizeof($v); $i++)
    {
        if ($v[$i] == 1)
        {
            print $i;

            if ($i != sizeof($v) - 1)
```

```
                echo ', ';
            }
        }

        echo "\n";
}

// Driver Code
block(71307);
block(1213);
block(29);
block(100);

// This code is contributed
// by Manish Shaw (manishshaw1)
?>
```

Output:

```
Blocks for 71307 : 0, 1, 3, 7, 9, 10, 12, 16
Blocks for 1213 : 0, 2, 3, 4, 5, 7, 10
Blocks for 29 : 0, 2, 3, 4
Blocks for 100 : 2, 5, 6
```

Improved By : andrew1234, jit_t, manishshaw1

Source

https://www.geeksforgeeks.org/powers-2-required-sum/

Chapter 254

Previous number same as 1's complement

Previous number same as 1's complement - GeeksforGeeks

Given a number check whether binary representation of its predecessor and its 1's complement are same or not.

Examples:

> Input : 14
> Output : NO
> Storing 14 as a 4 bit number, 14 (1110), its predecessor 13 (1101), its 1's complement 1 (0001), 13 and 1 are not same in their binary representation and hence output is NO.
>
> Input : 8
> Output : YES
> Storing 8 as a 4 bit number, 8 (1000), its predecessor 7 (0111), its 1's complement 7 (0111), both its predecessor and its 1's complement are 7 and hence output is YES.

Simple Approach: In this approach, we actually calculate the complement of the number.

1. Find binary representation of the number's predecessor and it's 1's complement using simple decimal to binary representation technique.

2. Compare bit by bit to check whether they are equal or not.

3. If all bits are equal then print YES else print NO.

Time Complexity: O (log n), as binary representation of numbers is getting calculated.

Auxiliary Space: O (1), although auxiliary space is O (1) still some memory spaces are getting
used to store binary representation of the numbers.

Efficient Approach: Only numbers which are powers of 2 have binary representation of their predecessor and their 1's complement as same.

1. Check whether a number is power of 2 or not.
2. If a number is power of 2 then print YES else print NO.

C++

```
 // An efficient C++ program to check if binary
// representations of n's predecessor and its
// 1's complement are same.
#include <bits/stdc++.h>
#define ull unsigned long long int
using namespace std;

// Returns true if binary representations of
// n's predecessor and it's 1's complement are same.
bool bit_check(ull n)
{
    if ((n & (n - 1)) == 0)
        return true;
    return false;
}

int main()
{
    ull n = 14;
    cout << bit_check(n) << endl;
    return 0;
}
```

Java

```
 // An efficient java program to check if binary
// representations of n's predecessor and its
// 1's complement are same.
public class GFG {

    // Returns true if binary representations of
    // n's predecessor and it's 1's complement
    // are same.
    static boolean bit_check(int n)
    {
        if ((n & (n - 1)) == 0)
            return true;
        return false;
    }
```

```
    // Driver code
    public static void main(String args[]) {

         int n = 14;
         if(bit_check(n))
            System.out.println ('1');
         else
            System.out.println('0');

    }
}

// This code is contributed by Sam007
```

C#

```
 // An efficient C# program to check if binary
// representations of n's predecessor and its
// 1's complement are same.
using System;
using System.Collections.Generic;

class GFG {

    // Returns true if binary representations of
    // n's predecessor and it's 1's complement
    // are same.
    static bool bit_check(int n)
    {
        if ((n & (n - 1)) == 0)
            return true;
        return false;
    }

    public static void Main()
    {
         int n = 14;
         if(bit_check(n))
            Console.WriteLine ('1');
         else
            Console.WriteLine ('0');
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// An efficient PHP program to check
// if binary representations of n's
// predecessor and its 1's complement
// are same.

// Returns true if binary
// representations of n's
// predecessor and it's 1's
// complement are same.
function bit_check($n)
{
    if (($n & ($n - 1)) == 0)
        return 1;
    return 0;
}

    // Driver code
    $n = 14;
    echo bit_check($n);

// This code is contributed by Sam007.
?>
```

Output:

```
0
```

Time Complexity: O (1)

Auxiliary Space : O (1) No extra space is getting used.

Improved By : Sam007

Source

https://www.geeksforgeeks.org/previous-number-1s-complement/

Chapter 255

Previous smaller integer having one less number of set bits

Previous smaller integer having one less number of set bits - GeeksforGeeks

Given a positive integer 'n' having 'x' number of set bits in its binary representation. The problem is to find the previous smaller integer(greatest integer smaller than n), having (x-1) number of set bits in its binary representation.

Note: 1 <= **n**

Examples :

```
Input : 8
Output : 0
(8)10 = (1000)2
is having 1 set bit.

(0)10 = (0)2
is having 0 set bit and is the previous smaller.

Input : 25
Output : 24
```

Following are the steps:

1. Find the position of the rightmost set bit(considering last bit at position 1, second last bit at position 2 and so on) in the binary representation of **n**. Let the position be represented by **pos**. Refer this post.
2. Turn off or unset the bit at position **pos**. Refer this post.

C++

```
 // C++ implementation to find the previous
// smaller integer with one less number of
// set bits
#include<bits/stdc++.h>
using namespace std;

// function to find the position of
// rightmost set bit.
int getFirstSetBitPos(int n)
{
    return log2(n & -n) + 1;
}

// function to find the previous smaller
// integer
int previousSmallerInteger(int n)
{
    // position of rightmost set bit of n
    int pos = getFirstSetBitPos(n);

    // turn off or unset the bit at
    // position 'pos'
    return (n & ~(1 << (pos - 1)));
}

// Driver program
int main()
{
    int n = 25;
    cout << previousSmallerInteger(n);
    return 0;
}
```

Java

```
 // Java implementation to find the previous
// smaller integer with one less number of
// set bits
class GFG {

    // function to find the position of
    // rightmost set bit.
    static int getFirstSetBitPos(int n)
    {
        return (int)(Math.log(n & -n) / Math.log(2)) + 1;
    }

    // function to find the previous smaller
```

```
    // integer
    static int previousSmallerInteger(int n)
    {

        // position of rightmost set bit of n
        int pos = getFirstSetBitPos(n);

        // turn off or unset the bit at
        // position 'pos'
        return (n & ~(1 << (pos - 1)));
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 25;
        System.out.print("Previous smaller Integer ="
                    + previousSmallerInteger(n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 implementation to find
# the previous smaller integer with
# one less number of set bits
import math

# Function to find the position
# of rightmost set bit.
def getFirstSetBitPos(n):

    return (int)(math.log(n & -n) /
                 math.log(2)) + 1

# Function to find the
# previous smaller integer
def previousSmallerInteger(n):

    # position of rightmost set bit of n
    pos = getFirstSetBitPos(n)

    # turn off or unset the bit
    # at position 'pos'
    return (n & ~(1 << (pos - 1)))
```

```
# Driver code
n = 25
print("Previous small Integer = ",
       previousSmallerInteger(n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# implementation to find the previous
// smaller integer with one less number of
// set bits
using System;

    class GFG {

    // function to find the position of
    // rightmost set bit.
    static int getFirstSetBitPos(int n)
    {
        return (int)(Math.Log(n & -n) /
                           Math.Log(2)) + 1;
    }

    // function to find the previous smaller
    // integer
    static int previousSmallerInteger(int n)
    {

        // position of rightmost set bit of n
        int pos = getFirstSetBitPos(n);

        // turn off or unset the bit at
        // position 'pos'
        return (n & ~(1 << (pos - 1)));
    }

    // Driver code
    public static void Main()
    {
        int n = 25;

        Console.WriteLine("Previous small Integer ="
                     + previousSmallerInteger(n));
    }
}

// This code is contributed by anant321.
```

PHP

```
 <?php
// PHP implementation to find the previous
// smaller integer with one less number of
// set bits

// function to find the position of
// rightmost set bit.

function getFirstSetBitPos($n)
{
    return log($n & -$n) + 1;
}

// function to find the previous
// smaller integer

function previousSmallerInteger($n)
{
    // position of rightmost set bit of n
    $pos = getFirstSetBitPos($n);

    // turn off or unset the bit at
    // position 'pos'
    return ($n & ~(1 << ($pos - 1)));
}

// Driver Code
$n = 25;
echo "Previous smaller Integer = ", previousSmallerInteger($n);

// This code is contributed by Ajit
?>
```

Output :

```
Previous smaller integer = 24
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/previous-smaller-integer-one-less-number-set-bits/

Chapter 256

Prime Number of Set Bits in Binary Representation | Set 1

Prime Number of Set Bits in Binary Representation | Set 1 - GeeksforGeeks

Given two integers 'L' and 'R', write a program to find the total numbers that are having prime number of set bits in their binary representation in the range [L, R].
Examples:

```
Input  : l = 6, r = 10
Output : 4
Explanation  :
6  -> 110  (2 set bits, 2 is prime)
7  -> 111  (3 set bits, 3 is prime)
9  -> 1001 (2 set bits, 2 is prime)
10 -> 1010 (2 set bits, 2 is prime)
Hence count is 4

Input  : l = 10, r = 15
Output : 5
10 -> 1010 (2 set bits, 2 is prime)
11 -> 1011 (3 set bits, 3 is prime)
12 -> 1100 (2 set bits, 2 is prime)
13 -> 1101 (3 set bits, 3 is prime)
14 -> 1110 (3 set bits, 3 is prime)
Hence count is 5
```

Explanation: In this program we find a total number, that's having prime number of set bit. so we use a CPP predefined function **___builtin__popcount()** these functions provide a total set bit in number. as well as be check the total bit's is prime or not if prime we increase the counter these process repeat till given range.

```
 // CPP program to count total prime
// number of set bits in given range
#include <bits/stdc++.h>
using namespace std;

bool isPrime(int n)
{
    // Corner cases
    if (n <= 1)  return false;
    if (n <= 3)  return true;

    // This is checked so that we can skip
    // middle five numbers in below loop
    if (n%2 == 0 || n%3 == 0) return false;

    for (int i=5; i*i<=n; i=i+6)
        if (n%i == 0 || n%(i+2) == 0)
           return false;

    return true;
}

// count number, that contains prime number of set bit
int primeBitsInRange(int l, int r)
{
    // tot_bit store number of bit in number
    int tot_bit, count = 0;

    // iterate loop from l to r
    for (int i = l; i <= r; i++) {

        // use predefined function for finding
        // set bit it is return number of set bit
        tot_bit = __builtin_popcount(i);

        // check tot_bit prime or, not
        if (isPrime(tot_bit))
            count++;
    }
    return count;
}

// Driven Program
int main()
{
    int l = 6, r = 10;
    cout << primeBitsInRange(l, r);
    return 0;
```

```
}
```

Output:

```
4
```

Time Complexity : Let's n = (r-l)
so overall time complexity is N*sqrt(N)

We can optimize above solution using Sieve of Eratosthenes.

Prime Number of Set Bits in Binary Representation | Set 2

Source

https://www.geeksforgeeks.org/prime-number-of-set-bits-in-binary-representation-set-1/

Chapter 257

Print all subsequences of a string | Iterative Method

Print all subsequences of a string | Iterative Method - GeeksforGeeks

Given a string s, print all possible subsequences of the given string in an iterative manner. We have already discussed **Recursive method to print all subsequences of a string**. Examples:

```
Input : abc
Output : a, b, c, ab, ac, bc, abc

Input : aab
Output : a, b, aa, ab, aab
```

Approah 1 :
Here, we discuss much easier and simpler iterative approach which is similar to Power Set. We use bit pattern from binary representation of 1 to 2^length(s) – 1.

input = "abc"
Binary representation to consider 1 to (2^3-1), i.e 1 to 7.
Start from left (MSB) to right (LSB) of binary representation and append characters from input string which corresponds to bit value 1 in binary representation to Final subsequence string sub.

Example:
001 => abc . Only c corresponds to bit 1. So, subsequence = c.
101 => abc . a and c corresponds to bit 1. So, subsequence = ac.

binary_representation (1) = 001 => c
binary_representation (2) = 010 => b
binary_representation (3) = 011 => bc
binary_representation (4) = 100 => a

binary_representation (5) = 101 => ac
binary_representation (6) = 110 => ab
binary_representation (7) = 111 => abc

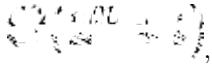
Below is the implementation of above approach:

```
 // CPP program to print all Subsequences
// of a string in iterative manner
#include <bits/stdc++.h>
using namespace std;

// function to find subsequence
string subsequence(string s, int binary, int len)
{
    string sub = "";
    for (int j = 0; j < len; j++)

        // check if jth bit in binary is 1
        if (binary & (1 << j))

            // if jth bit is 1, include it
            // in subsequence
            sub += s[j];

    return sub;
}

// function to print all subsequences
void possibleSubsequences(string s){

    // map to store subsequence
    // lexicographically by length
    map<int, set<string> > sorted_subsequence;

    int len = s.size();

    // Total number of non-empty subsequence
    // in string is 2^len-1
    int limit = pow(2, len);

    // i=0, corresponds to empty subsequence
    for (int i = 1; i <= limit - 1; i++) {

        // subsequence for binary pattern i
        string sub = subsequence(s, i, len);

        // storing sub in map
        sorted_subsequence[sub.length()].insert(sub);
    }
```

```
    for (auto it : sorted_subsequence) {

        // it.first is length of Subsequence
        // it.second is set<string>
        cout << "Subsequences of length = "
             << it.first << " are:" << endl;

        for (auto ii : it.second)

            // ii is iterator of type set<string>
            cout << ii << " ";

        cout << endl;
    }
}

// driver function
int main()
{
    string s = "aabc";
    possibleSubsequences(s);
    return 0;
}
```

Output:

```
Subsequences of length = 1 are:
a b c
Subsequences of length = 2 are:
aa ab ac bc
Subsequences of length = 3 are:
aab aac abc
Subsequences of length = 4 are:
aabc
```

Time Complexity :  , where n is length of string to find subsequences and l is length of binary string.

Approach 2 :
Approach is to get the position of rightmost set bit and and reset that bit after appending corresponding character from given string to the subsequence and will repeat the same thing till corresponding binary pattern has no set bits.

If input is s = “abc”
Binary representation to consider 1 to (2^3-1), i.e 1 to 7.

001 => abc . Only c corresponds to bit 1. So, subsequence = c
101 => abc . a and c corresponds to bit 1. So, subsequence = ac.

Let us use Binary representation of 5, i.e 101.
Rightmost bit is at position 1, append character at beginning of sub = c ,reset position 1 => 100
Rightmost bit is at position 3, append character at beginning of sub = ac ,reset position 3 => 000
As now we have no set bit left, we stop computing subsequence.

Example :

binary_representation (1) = 001 => c
binary_representation (2) = 010 => b
binary_representation (3) = 011 => bc
binary_representation (4) = 100 => a
binary_representation (5) = 101 => ac
binary_representation (6) = 110 => ab
binary_representation (7) = 111 => abc

Below is the implementation of above approach :

C++

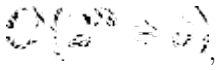
```
 // CPP code all Subsequences of a
// string in iterative manner
#include <bits/stdc++.h>
using namespace std;

// function to find subsequence
string subsequence(string s, int binary)
{
    string sub = "";
    int pos;

    // loop while binary is greater than 0
    while(binary>0)
    {
        // get the position of rightmost set bit
        pos=log2(binary&-binary)+1;

        // append at beginning as we are
        // going from LSB to MSB
        sub=s[pos-1]+sub;

        // resets bit at pos in binary
        binary= (binary & ~(1 << (pos-1)));
    }

    return sub;
```

```
}

// function to print all subsequences
void possibleSubsequences(string s){

    // map to store subsequence
    // lexicographically by length
    map<int, set<string> > sorted_subsequence;

    int len = s.size();

    // Total number of non-empty subsequence
    // in string is 2^len-1
    int limit = pow(2, len);

    // i=0, corresponds to empty subsequence
    for (int i = 1; i <= limit - 1; i++) {

        // subsequence for binary pattern i
        string sub = subsequence(s, i);

        // storing sub in map
        sorted_subsequence[sub.length()].insert(sub);
    }

    for (auto it : sorted_subsequence) {

        // it.first is length of Subsequence
        // it.second is set<string>
        cout << "Subsequences of length = "
            << it.first << " are:" << endl;

        for (auto ii : it.second)

            // ii is iterator of type set<string>
            cout << ii << " ";

        cout << endl;
    }
}

// driver function
int main()
{
    string s = "aabc";
    possibleSubsequences(s);

    return 0;
```

```
}
```

Output:

```
Subsequences of length = 1 are:
a b c
Subsequences of length = 2 are:
aa ab ac bc
Subsequences of length = 3 are:
aab aac abc
Subsequences of length = 4 are:
aabc
```

Time Complexity:  , where n is the length of string to find subsequence and b is the number of set bits in binary string.

Source

https://www.geeksforgeeks.org/print-subsequences-string-iterative-method/

Chapter 258

Print all the combinations of N elements by changing sign such that their sum is divisible by M

Print all the combinations of N elements by changing sign such that their sum is divisible by M - GeeksforGeeks

Given an array of N integers and an integer M. You can change the sign(positive or negative) of any element in the array. The task is to print all possible combinations of the array elements that can be obtained by changing the sign of the elements such that their sum is divisible by M.

Note: You have to take all of the array elements in each combination and in the same order as the elements present in the array. However, you can change the sign of elements.

Examples:

Input: a[] = {5, 6, 7}, M = 3
Output:
-5-6-7
+5-6+7
-5+6-7
+5+6+7

Input: a[] = {3, 5, 6, 8}, M = 5
Output:
-3-5+6-8
-3+5+6-8
+3-5-6+8
+3+5-6+8

Approach: The concept of power set is used here to solve this problem. Using power-set generate all possible combinations of signs that can be applied to the array of elements. If the sum obtained is divisible by M, then print the combination. Below are the steps:

- Iterate for all possible combinations of '+' and '-' using power set.
- Iterate on the array elements and if the j-th bit from left is set, then assume the array element to be positive and if the bit is not set, then assume the array element to be negative. Refer here to check if bit any index is set or not.
- If the sum is divisible by M, then the again traverse the array elements and print them along with sign('+' or '-').

Below is the implementation of the above approach:

C++

```
 #include <bits/stdc++.h>
using namespace std;

// Function to print all the combinations
void printCombinations(int a[], int n, int m)
{

    // Iterate for all combinations
    for (int i = 0; i < (1 << n); i++) {
        int sum = 0;

        // Initially 100 in binary if n is 3
        // as 1<<(3-1) = 100 in binary
        int num = 1 << (n - 1);

        // Iterate in the array and assign signs
        // to the array elements
        for (int j = 0; j < n; j++) {

            // If the j-th bit from left is set
            // take '+' sign
            if (i & num)
                sum += a[j];
            else
                sum += (-1 * a[j]);

            // Right shift to check if
            // jth bit is set or not
            num = num >> 1;
        }

        if (sum % m == 0) {

            // re-initialize
            num = 1 << (n - 1);
```

```
            // Iterate in the array elements
            for (int j = 0; j < n; j++) {

                // If the jth from left is set
                if ((i & num))
                    cout << "+" << a[j] << " ";
                else
                    cout << "-" << a[j] << " ";

                // right shift
                num = num >> 1;
            }
            cout << endl;
        }
    }
}

// Driver Code
int main()
{
    int a[] = { 3, 5, 6, 8 };
    int n = sizeof(a) / sizeof(a[0]);
    int m = 5;

    printCombinations(a, n, m);
    return 0;
}
```

Java

```
 import java.io.*;

class GFG
{

// Function to print
// all the combinations
static void printCombinations(int a[],
                              int n, int m)
{

    // Iterate for all
    // combinations
    for (int i = 0;
              i < (1 << n); i++)
    {
        int sum = 0;
```

```
// Initially 100 in binary
// if n is 3 as
// 1<<(3-1) = 100 in binary
int num = 1 << (n - 1);

// Iterate in the array
// and assign signs to
// the array elements
for (int j = 0; j < n; j++)
{

    // If the j-th bit
    // from left is set
    // take '+' sign
    if ((i & num) > 0)
        sum += a[j];
    else
        sum += (-1 * a[j]);

    // Right shift to check if
    // jth bit is set or not
    num = num >> 1;
}

if (sum % m == 0)
{

    // re-initialize
    num = 1 << (n - 1);

    // Iterate in the
    // array elements
    for (int j = 0; j < n; j++)
    {

        // If the jth from
        // left is set
        if ((i & num) > 0)
            System.out.print("+" +
                             a[j] + " ");
        else
            System.out.print("-" +
                             a[j] + " ");

        // right shift
        num = num >> 1;
    }
```

```
        System.out.println();
        }
    }
}

// Driver code
public static void main(String args[])
{
    int a[] = { 3, 5, 6, 8 };
    int n = a.length;
    int m = 5;

    printCombinations(a, n, m);
}
}

// This code is contributed
// by inder_verma.
```

Python3

```
 # Function to print
# all the combinations
def printCombinations(a, n, m):

    # Iterate for all
    # combinations
    for i in range(0, (1 << n)):

        sum = 0

        # Initially 100 in binary
        # if n is 3 as
        # 1<<(3-1) = 100 in binary
        num = 1 << (n - 1)

        # Iterate in the array
        # and assign signs to
        # the array elements
        for j in range(0, n):

            # If the j-th bit
            # from left is set
            # take '+' sign
            if ((i & num) > 0):
                sum += a[j]
            else:
                sum += (-1 * a[j])
```

```
            # Right shift to check if
            # jth bit is set or not
            num = num >> 1

        if (sum % m == 0):

            # re-initialize
            num = 1 << (n - 1)

            # Iterate in the
            # array elements
            for j in range(0, n):

                # If the jth from
                # left is set
                if ((i & num) > 0):
                    print("+", a[j], end = " ",
                                      sep = "")
                else:
                    print("-", a[j], end = " ",
                                      sep = "")

                # right shift
                num = num >> 1
            print("")

# Driver code
a = [ 3, 5, 6, 8 ]
n = len(a)
m = 5
printCombinations(a, n, m)

# This code is contributed
# by smita.
```

C#

```
 // Print all the combinations
// of N elements by changing
// sign such that their sum
// is divisible by M
using System;

class GFG
{

// Function to print
```

```
// all the combinations
static void printCombinations(int []a,
                              int n, int m)
{

    // Iterate for all
    // combinations
    for (int i = 0;
             i < (1 << n); i++)
    {
        int sum = 0;

        // Initially 100 in binary
        // if n is 3 as
        // 1<<(3-1) = 100 in binary
        int num = 1 << (n - 1);

        // Iterate in the array
        // and assign signs to
        // the array elements
        for (int j = 0; j < n; j++)
        {

            // If the j-th bit
            // from left is set
            // take '+' sign
            if ((i & num) > 0)
                sum += a[j];
            else
                sum += (-1 * a[j]);

            // Right shift to check if
            // jth bit is set or not
            num = num >> 1;
        }

        if (sum % m == 0)
        {

            // re-initialize
            num = 1 << (n - 1);

            // Iterate in the
            // array elements
            for (int j = 0; j < n; j++)
            {

                // If the jth from
```

```
                // left is set
                if ((i & num) > 0)
                    Console.Write("+" +
                              a[j] + " ");
                else
                    Console.Write("-" +
                              a[j] + " ");

                // right shift
                num = num >> 1;
            }

        Console.Write("\n");
        }
    }
}

// Driver code
public static void Main()
{
    int []a = { 3, 5, 6, 8 };
    int n = a.Length;
    int m = 5;

    printCombinations(a, n, m);
}
}

// This code is contributed
// by Smitha.
```

Output:

```
-3 -5 +6 -8
-3 +5 +6 -8
+3 -5 -6 +8
+3 +5 -6 +8
```

Time Complexity: O(2^N * N), where N is the number of elements.

Improved By : inderDuMCA, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/print-all-the-combinations-of-n-elements-by-changing-sign-such-that-their-sum-is-d

Chapter 259

Print bitwise AND set of a number N

Print bitwise AND set of a number N - GeeksforGeeks

Given a number N, print all the numbers which are a bitwise AND set of the binary representation of N. Bitwise AND set of a number N is all possible numbers x smaller than or equal N such that N & i is equal to x for some number i.

Examples :

```
Input : N = 5
Output : 0, 1, 4, 5
Explanation: 0 & 5 = 0
             1 & 5 = 1
             2 & 5 = 0
             3 & 5 = 1
             4 & 5 = 4
             5 & 5 = 5
So we get 0, 1, 4 and 5 in the
bitwise subsets of N.

Input : N = 9
Output : 0, 1, 8, 9
```

Simple Approach: A naive approach is to iterate from all numbers from 0 to N and check if (N&i == i). Print the numbers which satisfy the specified condition.

Below is the implementation of above idea:

C++

```
 // CPP program to print all bitwise
// subsets of N (Naive approach)
#include <bits/stdc++.h>
using namespace std;

// function to find bitwise subsets
// Naive approach
void printSubsets(int n) {
  for (int i = 0; i <= n; i++)
    if ((n & i) == i)
      cout << i << " ";
}

// Driver Code
int main() {

  int n = 9;
  printSubsets(n);
  return 0;
}
```

Java

```
 // JAVA program to print all bitwise
// subsets of N (Naive approach)
class GFG {

    // function to find bitwise subsets
    // Naive approach
    static void printSubsets(int n)
    {

        for (int i = 0; i <= n; i++)
            if ((n & i) == i)
                System.out.print(i + " ");
    }

    // Driver function
    public static void main(String[] args)
    {
        int n = 9;

        printSubsets(n);
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python program to print all bitwise
# subsets of N (Naive approach)
def printSubsets(n):

    for i in range(n + 1):

        if ((n & i) == i):
            print(i ," ", end = "")

# Driver code
n = 9
printSubsets(n)

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to print all bitwise
// subsets of N (Naive approach)
using System;

class GFG {

    // function to find bitwise subsets
    // Naive approach
    static void printSubsets(int n)
    {

        for (int i = 0; i <= n; i++)
            if ((n & i) == i)
                Console.Write(i + " ");
    }

    // Driver function
    public static void Main()
    {
        int n = 9;

        printSubsets(n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to print all bitwise
// subsets of N (Naive approach)

// function to find bitwise subsets
// Naive approach
function printSubsets($n)
{
for ($i = 0; $i <= $n; $i++)
    if (($n & $i) == $i)
    echo $i." ";
}

// Driver Code
$n = 9;
printSubsets($n);

// This code is contributed by mits
?>
```

Output :

```
0 1 8 9
```

Time Complexity : O(N)

Efficient Solution: An efficient solution is to use bitwise operators to find the subsets. Instead of iterating for every i, we can simply iterate for the bitwise subsets only. Iterating backward for **i=(i-1)&n** gives us every bitwise subset, where i starts from n and ends at 1.

Below is the implementation of above idea:

CPP

```
 // CPP program to print all bitwise
// subsets of N (Efficient approach)

#include <bits/stdc++.h>
using namespace std;

// fucntion to find bitwise subsets
// Efficient approach
void printSubsets(int n) {

  for (int i = n; i > 0; i = (i - 1) & n)
    cout << i << " ";
  cout << 0;
}
```

```
// Driver Code
int main() {
  int n = 9;
  printSubsets(n);
  return 0;
}
```

Java

```
 // Java program to print all bitwise
// subsets of N (Efficient approach)

class GFG
{

    // function to find bitwise
    // subsets Efficient approach
    static void printSubsets(int n)
    {
    for (int i = n; i > 0; i = (i - 1) & n)

        System.out.print(i + " ");
        System.out.print(" 0 ");

    }

// Driver Code
public static void main(String[] args)
{
    int n = 9;
    printSubsets(n);
}
}

// This code is contributed by ajit.
```

Python3

```
 # Python 3 program to
# print all bitwise
# subsets of N
# (Efficient approach)

# fucntion to find
# bitwise subsets
# Efficient approach
```

```
def printSubsets(n):
    i=n
    while(i != 0):
        print(i,end=" ")
        i=(i - 1) & n
    print("0")

# Driver Code
n = 9
printSubsets(n)

# This code is contributed by
# Smith Dinesh Semwal
```

C#

```
 // C# program to print all bitwise
// subsets of N (Efficient approach)
using System;

public class GFG {

    // fucntion to find bitwise subsets
    // Efficient approach
    static void printSubsets(int n) {

    for (int i = n; i > 0; i = (i - 1) & n)
        Console.Write(i +" ");
        Console.WriteLine("0");
    }

    // Driver Code
    static public void Main () {

        int n = 9;

        printSubsets(n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to print all bitwise
// subsets of N (Efficient approach)
```

```
// function to find bitwise subsets
// Efficient approach
function printSubsets($n)
{

    for ($i = $n; $i > 0;
         $i = ($i - 1) & $n)

        echo $i." ";
    echo "0";
}

// Driver Code
$n = 9;
printSubsets($n);

// This code is contributed by mits
?>
```

Output :

```
9 8 1 0
```

Time Complexity: O(K), where K is the number of bitwise subsets of N.

Improved By : Mithun Kumar, jit_t

Source

https://www.geeksforgeeks.org/print-bitwise-and-set-of-a-number-n/

Chapter 260

Print first n numbers with exactly two set bits

Print first n numbers with exactly two set bits - GeeksforGeeks

Given a number n, print first n positive integers with exactly two set bits in their binary representation.

Examples :

```
Input: n = 3
Output: 3 5 6
The first 3 numbers with two set bits are 3 (0011),
5 (0101) and 6 (0110)

Input: n = 5
Output: 3 5 6 9 10 12
```

A **Simple Solution** is to consider all positive integers one by one starting from 1. For every number, check if it has exactly two sets bits. If a number has exactly two set bits, print it and increment count of such numbers.

An **Efficient Solution** is to directly generate such numbers. If we clearly observe the numbers, we can rewrite them as given below pow(2,1)+pow(2,0), pow(2,2)+pow(2,0), pow(2,2)+pow(2,1), pow(2,3)+pow(2,0), pow(2,3)+pow(2,1), pow(2,3)+pow(2,2), ……..

All numbers can be generated in increasing order according to higher of two set bits. The idea is to fix higher of two bits one by one. For current higher set bit, consider all lower bits and print the formed numbers.

C++

```
// C++ program to print first n numbers
```

```
// with exactly two set bits
#include <iostream>
using namespace std;

// Prints first n numbers with two set bits
void printTwoSetBitNums(int n)
{
    // Initialize higher of two sets bits
    int x = 1;

    // Keep reducing n for every number
    // with two set bits.
    while (n > 0)
    {
        // Consider all lower set bits for
        // current higher set bit
        int y = 0;
        while (y < x)
        {
            // Print current number
            cout << (1 << x) + (1 << y) << " ";

            // If we have found n numbers
            n--;
            if (n == 0)
                return;

            // Consider next lower bit for current
            // higher bit.
            y++;
        }

        // Increment higher set bit
        x++;
    }
}

// Driver code
int main()
{
    printTwoSetBitNums(4);

    return 0;
}
```

Java

```
 // Java program to print first n numbers
```

```
// with exactly two set bits
import java.io.*;

class GFG
{
    // Function to print first n numbers with two set bits
    static void printTwoSetBitNums(int n)
    {
        // Initialize higher of two sets bits
        int x = 1;

        // Keep reducing n for every number
        // with two set bits
        while (n > 0)
        {
            // Consider all lower set bits for
            // current higher set bit
            int y = 0;
            while (y < x)
            {
                // Print current number
                System.out.print(((1 << x) + (1 << y)) +" ");

                // If we have found n numbers
                n--;
                if (n == 0)
                    return;

                // Consider next lower bit for current
                // higher bit.
                y++;
            }

            // Increment higher set bit
            x++;
        }
    }

    // Driver program
    public static void main (String[] args)
    {
        int n = 4;
        printTwoSetBitNums(n);
    }
}

// This code is contributed by Pramod Kumar
```

C#

```
 // C# program to print first n numbers
// with exactly two set bits
using System;

class GFG
    {

    // Function to print first n
    // numbers with two set bits
    static void printTwoSetBitNums(int n)
    {

        // Initialize higher of
        // two sets bits
        int x = 1;

        // Keep reducing n for every
        // number with two set bits
        while (n > 0)
        {

            // Consider all lower set bits
            // for current higher set bit
            int y = 0;
            while (y < x)
            {

                // Print current number
                Console.Write(((1 << x) +
                               (1 << y)) +" ");

                // If we have found n numbers
                n--;
                if (n == 0)
                    return;

                // Consider next lower bit
                // for current higher bit.
                y++;
            }

            // Increment higher set bit
            x++;
        }
    }
```

```
    // Driver program
    public static void Main()
    {
        int n = 4;
        printTwoSetBitNums(n);
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program to print
// first n numbers with
// exactly two set bits

// Prints first n numbers
// with two set bits
function printTwoSetBitNums($n)
{
    // Initialize higher of
    // two sets bits
    $x = 1;

    // Keep reducing n for
    // every number with
    // two set bits.
    while ($n > 0)
    {
        // Consider all lower set
        // bits for current higher
        // set bit
        $y = 0;
        while ($y < $x)
        {
            // Print current number
            echo (1 << $x) + (1 << $y), " ";

            // If we have found n numbers
            $n--;
            if ($n == 0)
                return;

            // Consider next lower
            // bit for current
            // higher bit.
            $y++;
```

```
        }

        // Increment higher set bit
        $x++;
    }
}

// Driver code
printTwoSetBitNums(4);

// This code is contributed by Ajit
?>
```

Output :

```
3 5 6 9
```

Time Complexity : O(n)

Improved By : jit_t

Source

https://www.geeksforgeeks.org/print-first-n-numbers-with-exactly-two-set-bits/

Chapter 261

Print numbers having first and last bits as the only set bits

Print numbers having first and last bits as the only set bits - GeeksforGeeks

Given a positive integer **n**. The problem is to print numbers in the range 1 to n having first and last bits as the only set bits.

Examples:

```
Input : n = 10
Output : 1 3 5 9
(1)10 = (1)2.
(3)10 = (11)2.
(5)10 = (101)2.
(9)10 = (1001)2
```

Naive Approach: Print "1". Now for **i** = 3 to n, check if **(i-1)** is a Perfect power of two or not. If true then print **i**.

C++

```
 // C++ implementation to print numbers in the range 1 to n
// having first and last bits as the only set bits
#include <bits/stdc++.h>

using namespace std;

typedef unsigned long long int ull;

// function to check whether 'n'
// is a power of 2 or not
```

```
bool powerOfTwo(ull n)
{
    return (!(n & n-1));
}

// fuction to print numbers in the range 1 to n having
// first and last bits as the only set bits
void printNumWithFirstLastBitsSet(ull n)
{
    ull i = 1;

    // first number is '1'
    cout << i << " ";

    // generating all the numbers
    for (i = 3; i <= n; i++)
        // if true, then print 'i'
        if (powerOfTwo(i-1))
            cout << i << " ";
}

// Driver program to test above
int main()
{
    ull n = 10;
    printNumWithFirstLastBitsSet(n);
    return 0;
}
```

Java

```
 // Naive approach
// Java implementation to print
// numbers in the range 1 to n
// having first and last bits as
// the only set bits
import java.io.*;

class GFG {

    // function to check whether 'n'
    // is a power of 2 or not
    static Boolean powerOfTwo(long n)
    {
        return (!((n & n-1) != 0));
    }

    // fuction to print numbers in the
```

```
    // range 1 to n having first and
    // last bits as the only set bits
    static void printNumWithFirstLastBitsSet(long n)
    {
        long i = 1;

        // first number is '1'
        System.out.print( i + " ");

        // generating all the numbers
        for (i = 3; i <= n; i++)

            // if true, then print 'i'
            if (powerOfTwo(i - 1))
                System.out.print(i + " ");
    }
        // Driver function
        public static void main (String[] args) {
        long n = 10l;
        printNumWithFirstLastBitsSet(n);

        }
}

//This code is contributed by Gitanjali.
```

Python3

```
 # Python implementation to print
# numbers in the range 1 to n
# having first and last bits
# as the only set bits
import math

# function to check whether 'n'
# is a power of 2 or not
def powerOfTwo(n):
    re = (n & n - 1)
    return (re == 0)

# fuction to print numbers
# in the range 1 to n having
# first and last bits as
# the only set bits
def printNumWithFirstLastBitsSet(n):
    i = 1

    # first number is '1'
```

```
    print ( i, end = " ")

    # generating all the numbers
    for i in range(3, n + 1):

        # if true, then print 'i'
        if (powerOfTwo(i - 1)):
            print ( i, end = " ")


# driver function
n = 10
printNumWithFirstLastBitsSet(n)

# This code is contributed by Gitanjali.
```

C#

```
 // Naive approach
// C# implementation to print
// numbers in the range 1 to n
// having first and last bits as
// the only set bits
using System;

class GFG {

    // function to check whether 'n'
    // is a power of 2 or not
    static Boolean powerOfTwo(long n)
    {
        return (!((n & n-1) != 0));
    }

    // fuction to print numbers in the
    // range 1 to n having first and
    // last bits as the only set bits
    static void printNumWithFirstLastBitsSet(long n)
    {
        long i = 1;

        // first number is '1'
        Console.Write( i + " ");

        // generating all the numbers
        for (i = 3; i <= n; i++)

            // if true, then print 'i'
```

```
            if (powerOfTwo(i - 1))
                Console.Write(i + " ");
    }

    // Driver function
    public static void Main ()
    {
        long n = 10L;
        printNumWithFirstLastBitsSet(n);

    }
}

// This code is contributed by Vt_m.
```

PHP

```
 <?php
// php implementation to print
// numbers in the range 1 to n
// having first and last bits
// as the only set bits

// function to check whether 'n'
// is a power of 2 or not
function powerOfTwo($n)
{
    return (!($n & $n - 1));
}

// fuction to print numbers in
// the range 1 to n having
// first and last bits as
// the only set bits
function printNumWithFirstLastBitsSet($n)
{
    $i = 1;

    // first number is '1'
    echo $i." ";

    // generating all the numbers
    for ($i = 3; $i <= $n; $i++)

        // if true, then print 'i'
        if (powerOfTwo($i - 1))
            echo $i." ";
}
```

```
    // Driver Code
    $n = 10;
    printNumWithFirstLastBitsSet($n);

// This Code is contributed by mits
?>
```

Output:

```
1 3 5 9
```

Time Complexity: O(n).

Efficient Approach: Print "1". Now one by one generate perfect power of two (except '1') with the help of bitwise left shift operation. Bitwise xor these numbers with 1 and if result is in the range print them else stop.

C++

```
 // C++ implementation to print numbers in the range 1 to n
// having first and last bits as the only set bits
#include <bits/stdc++.h>

using namespace std;

typedef unsigned long long int ull;

// fuction to print numbers in the range 1 to n having
// first and last bits as the only set bits
void printNumWithFirstLastBitsSet(ull n)
{
    ull power_2 = 1, num;

    // first number is '1'
    cout << power_2 << " ";

    while (1)
    {
        // obtaining next perfect power of 2
        power_2 <<= 1;

        // toggling the last bit to convert
        // it to as set bit
        num = power_2 ^ 1;

        // if out of range then break;
```

```
        if (n < num)
            break;

        // display
        cout << num << " ";
    }
}

// Driver program to test above
int main()
{
    ull n = 10;
    printNumWithFirstLastBitsSet(n);
    return 0;
}
```

Java

```
 // efficient approach Java implementation
// to print numbers in the range 1 to n
// having first and last bits as the only set bits

import java.io.*;

class GFG {

    // fuction to print numbers in
    // the range 1 to n having first and
    // last bits as the only set bits
    static void prNumWithFirstLastBitsSet(long n)
    {
        long power_2 = 1, num;

        // first number is '1'
        System.out.print(power_2 + " ");

        while (true)
        {
            // obtaining next perfect power of 2
            power_2 <<= 1;

            // toggling the last bit to
            // convert it to as set bit
            num = power_2 ^ 1;

            // if out of range then break;
            if (n < num)
                break;
```

```
            // display
            System.out.print(num + " ");
        }

}

    public static void main (String[] args) {
    long n = 10;
    prNumWithFirstLastBitsSet(n);

    }
}
// This code is contributed by Gitanjali.
```

Python3

```
 # Python3 implementation to
# pr numbers in the range
# 1 to n having first and
# last bits as the only set bits


# fuction to pr numbers in the
# range 1 to n having first and
# last bits as the only set bits
def prNumWithFirstLastBitsSet(n):

    power_2 = 1

    # first number is '1'
    print ( power_2, end = ' ')

    while (1):
        # obtaining next perfect
        # power of 2
        power_2 <<= 1

        # toggling the last bit to
        # convert it to as set bit
        num = power_2 ^ 1

        # if out of range then break;
        if (n < num):
            break

        # display
        print ( num, end = ' ')
```

```
# Driver program
n = 10;
prNumWithFirstLastBitsSet(n)

# This code is contributed by saloni1297
```

C#

```
 // efficient approach C# implementation
// to print numbers in the range 1 to n
// having first and last bits as the only set bits
using System;

class GFG {

    // fuction to print numbers in
    // the range 1 to n having first and
    // last bits as the only set bits
    static void prNumWithFirstLastBitsSet(long n)
    {
        long power_2 = 1, num;

        // first number is '1'
        Console.Write(power_2 + " ");

        while (true)
        {
            // obtaining next perfect power of 2
            power_2 <<= 1;

            // toggling the last bit to
            // convert it to as set bit
            num = power_2 ^ 1;

            // if out of range then break;
            if (n < num)
                break;

            // display
            Console.Write(num + " ");
        }

    }

    // Driver code
```

```
    public static void Main ()
    {
        long n = 10;
        prNumWithFirstLastBitsSet(n);

    }
}
// This code is contributed by vt_m.
```

PHP

```
 <?php
// php implementation to print
// numbers in the range 1 to n
// having first and last bits
// as the only set bits

// fuction to print numbers in
// the range 1 to n having
// first and last bits as
// the only set bits
function printNumWithFirstLastBitsSet($n)
{
    $power_2 = 1;

    // first number is '1'
    echo $power_2." ";

    while (1)
    {

        // obtaining next perfect
        // power of 2
        $power_2 <<= 1;

        // toggling the last
        // bit to convert
        // it to as set bit
        $num = $power_2 ^ 1;

        // if out of range
        // then break;
        if ($n < $num)
            break;

        // display
        echo $num." ";
    }
```

```
}

    // Driver code
    $n = 10;
    printNumWithFirstLastBitsSet($n);

// This code is contributed by mits
?>
```

Output:

```
1 3 5 9
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/print-numbers-first-last-bits-set-bits/

Chapter 262

Print numbers in the range 1 to n having bits in alternate pattern

Print numbers in the range 1 to n having bits in alternate pattern - GeeksforGeeks

Given a positive integer **n**. The problem is to print the numbers in the range 1 to n having bits in alternate pattern. Here alternate pattern means that the set and unset bits in the number occur in alternate order. For example- 5 has an alternate pattern i.e. 101.

Examples:

```
Input : n = 10
Output : 1 2 5 10

Input : n = 50
Output : 1 2 5 10 21 42
```

Method 1 (Naive Approach): Generate all the numbers in the range 1 to n and for each generated number check whether it has bits in alternate pattern. Time Complexity is of O(n).

Method 2 (Efficient Approach): Algorithm:

```
printNumHavingAltBitPatrn(n)
    Initialize curr_num = 1
    print curr_num
    while (1)
        curr_num <<= 1
    if n < curr_num then
```

```
        break
    print curr_num
    curr_num = ((curr_num) << 1) ^ 1
    if n < curr_num then
        break
    print curr_num
```

CPP

```
 // C++ implementation to print numbers in the range
// 1 to n having bits in alternate pattern
#include <bits/stdc++.h>

using namespace std;

// function to print numbers in the range 1 to n
// having bits in alternate pattern
void printNumHavingAltBitPatrn(int n)
{
    // first number having bits in alternate pattern
    int curr_num = 1;

    // display
    cout << curr_num << " ";

    // loop until n < curr_num
    while (1) {

        // generate next number having alternate
        // bit pattern
        curr_num <<= 1;

        // if true then break
        if (n < curr_num)
            break;

        // display
        cout << curr_num << " ";

        // generate next number having alternate
        // bit pattern
        curr_num = ((curr_num) << 1) ^ 1;

        // if true then break
        if (n < curr_num)
            break;

        // display
```

```
        cout << curr_num << " ";
    }
}

// Driver program to test above
int main()
{
    int n = 50;
    printNumHavingAltBitPatrn(n);
    return 0;
}
```

Java

```
 // Java implementation to print numbers in the range
// 1 to n having bits in alternate pattern

import java.io.*;
import java.util.*;

class GFG
{
    public static void printNumHavingAltBitPatrn(int n)
    {
        // first number having bits in alternate pattern
        int curr_num = 1, i = 1;

        // display
        System.out.print(curr_num + " ");

        // loop until n < curr_num
        while (i!=0)
        {
            i++;
            // generate next number having alternate
            // bit pattern
            curr_num <<= 1;

            // if true then break
            if (n < curr_num)
                break;

            // display
            System.out.print(curr_num + " ");

            // generate next number having alternate
            // bit pattern
            curr_num = ((curr_num) << 1) ^ 1;
```

```
            // if true then break
            if (n < curr_num)
                break;

            // display
            System.out.print(curr_num + " ");
        }
    }
    public static void main (String[] args)
    {
        int n = 50;
        printNumHavingAltBitPatrn(n);
    }
}

// Code Contributed by Mohit Gupta_OMG <(0_o)>
```

Python3

```
 # Python3 program for count total
# zero in product of array

# function to print numbers in the range
# 1 to nhaving bits in alternate pattern
def printNumHavingAltBitPatrn(n):

    # first number having bits in
    # alternate pattern
    curr_num = 1

    # display
    print (curr_num)

    # loop until n < curr_num
    while (1) :

        # generate next number having
        # alternate bit pattern
        curr_num = curr_num << 1;

        # if true then break
        if (n < curr_num):
            break;

        # display
        print( curr_num )
```

```
        # generate next number having
        # alternate bit pattern
        curr_num = ((curr_num) << 1) ^ 1;

        # if true then break
        if (n < curr_num):
            break

        # display
        print( curr_num )

# Driven code
n = 50
printNumHavingAltBitPatrn(n)

# This code is contributed by "rishabh_jain".
```

C#

```
 // C# implementation to print numbers in the range
// 1 to n having bits in alternate pattern
using System;

class GFG {

    // function to print numbers in the range 1 to n
    // having bits in alternate pattern
    public static void printNumHavingAltBitPatrn(int n)
    {

        // first number having bits in alternate pattern
        int curr_num = 1, i = 1;

        // display
        Console.Write(curr_num + " ");

        // loop until n < curr_num
        while (i!=0)
        {

            // generate next number having alternate
            // bit pattern
            curr_num <<= 1;

            // if true then break
            if (n < curr_num)
                break;
```

```
            // display
            Console.Write(curr_num + " ");

            // generate next number having alternate
            // bit pattern
            curr_num = ((curr_num) << 1) ^ 1;

            // if true then break
            if (n < curr_num)
                break;

            // display
            Console.Write(curr_num + " ");
        }
    }

    // Driver code
    public static void Main ()
    {
        int n = 50;

        printNumHavingAltBitPatrn(n);
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// php implementation to print
// numbers in the range
// 1 to n having bits in
// alternate pattern


// function to print numbers
// in the range 1 to n
// having bits in alternate
// pattern
function printNumHavingAltBitPatrn($n)
{

    // first number having bits
    // in alternate pattern
    $curr_num = 1;

    // display
```

```
    echo $curr_num." ";

    // loop until n < curr_num
    while (1)
    {

        // generate next number
        // having alternate
        // bit pattern
        $curr_num <<= 1;

        // if true then break
        if ($n < $curr_num)
            break;

        // display
        echo $curr_num." ";

        // generate next number
        // having alternate
        // bit pattern
        $curr_num = (($curr_num) << 1) ^ 1;

        // if true then break
        if ($n < $curr_num)
            break;

        // display
        echo $curr_num." ";
    }
}

    // Driver code
    $n = 50;
    printNumHavingAltBitPatrn($n);

// This code is contributed by mits
?>
```

Output:

```
1 2 5 10 21 42
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/print-numbers-range-1-n-bits-alternate-pattern/

Chapter 263

Print pair with maximum AND value in an array

Print pair with maximum AND value in an array - GeeksforGeeks

Given an array of n positive elements, find the maximum AND value and the pair of elements generating the maximum AND value from the array.
AND is bitwise & operator.

Examples:

```
Input : arr[] = {4, 8, 12, 16}
Output : Pair = 8, 12
         Maximum AND value = 8

Input : arr[] = {4, 8, 16, 2}
Output : Pair = Not Possible
         Maximum AND value = 0
```

Approach:
Finding Maximum AND value is same as Maximum AND value in an array. Our task is to find the pair of elements resulting in obtained AND value. For finding the elements, simply traverse the whole array and find the AND value of each element with the obtained maximum AND value (result) and if **arr[i] & result == result** , that means arr[i] is the element which will generate maximum AND value. Also, in the case if maximum AND value (result) is zero then we should print "Not possible" in that case.

Below is the implementation of above approach:

C++

```
// CPP Program to find pair with
```

```
// maximum AND value
#include <bits/stdc++.h>
using namespace std;

// Utility function to check number of
// elements having set msb as of pattern
int checkBit(int pattern, int arr[], int n)
{
    int count = 0;
    for (int i = 0; i < n; i++)
        if ((pattern & arr[i]) == pattern)
            count++;
    return count;
}

// Function for finding maximum and
// value pair
int maxAND(int arr[], int n)
{
    int res = 0, count;

    // iterate over total of 30bits
    // from msb to lsb
    for (int bit = 31; bit >= 0; bit--) {

        // find the count of element
        // having set msb
        count = checkBit(res | (1 << bit), arr, n);

        // if count >= 2 set particular
        // bit in result
        if (count >= 2)
            res |= (1 << bit);
    }

    // Find the elements
    if (res == 0)
        cout << "Not Possible\n";

    else {

        // print the pair of elements
        cout << "Pair = ";

        count = 0;

        for (int i = 0; i < n && count < 2; i++) {
```

```
            // inc count value after
            // printing element
            if ((arr[i] & res) == res) {
                count++;
                cout << arr[i] << " ";
            }
        }
    }

    // return the result value
    return res;
}

// Driver function
int main()
{
    int arr[] = { 4, 8, 6, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "\nMaximum AND Value = "
         << maxAND(arr, n);
    return 0;
}
```

Java

```
 // Java Program to find pair
// with maximum AND value
import java.util.*;
import java.io.*;

class GFG
{
// Utility function to check number of
// elements having set msb as of pattern
static int checkBit(int pattern, int arr[], int n)
{
    int count = 0;

    for (int i = 0; i < n; i++)
        if ((pattern & arr[i]) == pattern)
            count++;

    return count;
}

// Function for finding maximum and
// value pair
static int maxAND(int arr[], int n)
```

```
{
    int res = 0, count;

    // iterate over total of 30bits
    // from msb to lsb
    for (int bit = 31; bit >= 0; bit--) {

        // find the count of element
        // having set msb
        count = checkBit(res | (1 << bit), arr, n);

        // if count >= 2 set particular
        // bit in result
        if (count >= 2)
            res |= (1 << bit);
    }

    // Find the elements
    if (res == 0)
        System.out.println("Not Possible");

    else {

        // print the pair of elements
        System.out.print("Pair = ");

        count = 0;

        for (int i = 0; i < n && count < 2; i++) {

            // inc count value after
            // printing element
            if ((arr[i] & res) == res) {
                count++;
                System.out.print(arr[i] + " ");
            }
        }
        System.out.println();
    }

    // return the result value
    return res;
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 4, 8, 6, 2 };
```

```
    int n = arr.length;
    System.out.println("Maximum AND Value = "
                                + maxAND(arr, n));

}
}

// This code is contributed by Sahil_Bansall
```

Python 3

```
 # Python 3 Program to find pair with
# maximum AND value

# Utility function to check number of
# elements having set msb as of pattern
def checkBit(pattern, arr, n):

    count = 0
    for i in range(0, n):
        if ((pattern & arr[i]) == pattern):
            count += 1
    return count

# Function for finding maximum and
# value pair
def maxAND(arr, n):

    res = 0

    # iterate over total of 30bits
    # from msb to lsb
    for bit in range(31 ,-1 ,-1) :

        # find the count of element
        # having set msb
        count = checkBit(res | (1 << bit),
                                    arr, n)

        # if count >= 2 set particular
        # bit in result
        if (count >= 2):
            res |= (1 << bit)

    # Find the elements
    if (res == 0):
        print("Not Possible")
```

```
    else:
        # print the pair of elements
        print("Pair = ", end = "")

        count = 0

        i = 0
        while(i < n and count < 2):

            # inc count value after
            # printing element
            if ((arr[i] & res) == res) :
                count+=1
                print(arr[i] , end = " ")
            i += 1

    # return the result value
    return res

# Driver function
arr = [4, 8, 6, 2 ]
n = len(arr)
print("\nMaximum AND Value = ",
                    maxAND(arr, n))

# This code is contributed by Smitha
```

C#

```
 // C# Program to find pair
// with maximum AND value
using System;

class GFG
    {
    // Utility function to check number of
    // elements having set msb as of pattern
    static int checkBit(int pattern, int []arr, int n)
    {
        int count = 0;

        for (int i = 0; i < n; i++)
            if ((pattern & arr[i]) == pattern)
                count++;

        return count;
    }
```

```
// Function for finding maximum and
// value pair
static int maxAND(int []arr, int n)
{
    int res = 0, count;

    // iterate over total of 30bits
    // from msb to lsb
    for (int bit = 31; bit >= 0; bit--) {

        // find the count of element
        // having set msb
        count = checkBit(res | (1 << bit), arr, n);

        // if count >= 2 set particular
        // bit in result
        if (count >= 2)
            res |= (1 << bit);
    }

    // Find the elements
    if (res == 0)
        Console.Write("Not Possible");

    else {

        // print the pair of elements
        Console.Write("Pair = ");

        count = 0;

        for (int i = 0; i < n && count < 2; i++)
        {

            // inc count value after
            // printing element
            if ((arr[i] & res) == res) {
                count++;
                Console.Write(arr[i] + " ");
            }
        }
        Console.WriteLine();
    }

    // return the result value
    return res;
}
```

```
    // Driver code
    public static void Main()
    {
        int []arr = { 4, 8, 6, 2 };
        int n = arr.Length;
        Console.WriteLine("Maximum AND Value = "
                                    + maxAND(arr, n));

    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// php Program to find pair with
// maximum AND value

// Utility function to check number of
// elements having set msb as of pattern
function checkBit($pattern, $arr, $n)
{
    $count = 0;
    for ($i = 0; $i < $n; $i++)
        if (($pattern & $arr[$i]) == $pattern)
            $count++;
    return $count;
}

// Function for finding maximum
// and  value pair
function maxAND($arr, $n)
{
    $res = 0;

    // iterate over total of 30bits
    // from msb to lsb
    for ($bit = 31; $bit >= 0; $bit--)
    {

        // find the count of element
        // having set msb
        $count = checkBit($res | (1 << $bit),
                                    $arr, $n);

        // if count >= 2 set particular
        // bit in result
```

```
        if ($count >= 2)
            $res |= (1 << $bit);
    }

    // Find the elements
    if ($res == 0)
        echo "Not Possible\n";

    else {

        // print the pair of elements
        echo "Pair = ";

        $count = 0;

        for ($i = 0; $i < $n &&
             $count < 2; $i++)
        {

            // inc count value after
            // printing element
            if (($arr[$i] & $res) == $res)
            {
                $count++;
                echo $arr[$i]. " ";
            }
        }
    }

    // return the result value
    return $res;
}

    // Driver code
    $arr = array( 4, 8, 6, 2 );
    $n = sizeof($arr) / sizeof($arr[0]);
    echo "\nMaximum AND Value = " .maxAND($arr, $n);

//This code is contributed by mits
?>
```

Output:

```
Pair = 4 6
Maximum AND value = 4
```

Improved By : Mithun Kumar, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/print-pair-with-maximum-and-value-in-an-array/

Chapter 264

Print ‘K’th least significant bit of a number

Print ’K’th least significant bit of a number - GeeksforGeeks

A number N is given. We need to print its ‘K’th Least Significant Bit.

Examples :

```
Input : num = 10, k = 4
Output : 1
Explanation : Binary Representation
of 10 is 1010. 4th LSB is 1.

Input : num = 16, k = 3
Output : 0
Explanation : Binary Representation
of 16 is 10000. 3rd LSB is 0.
```

We can easily solve this problem by following steps :

1. Shift the number ‘1’ (K-1) times left.
2. This will yield a number with all unset bits but the ‘K’th bit. Now, we’ll perform logical AND of the shifted number with given number.
3. All bits except the ‘K’th bit will yield 0, and ‘K’th bit will depend on the number. This is because, 1 AND 1 is 1. 0 AND 1 is 0.

C++

```
 // CPP code to print 'K'th LSB
#include <bits/stdc++.h>
```

```
using namespace std;

//Function returns 1 if set, 0 if not
bool LSB(int num, int K)
{
    return (num & (1 << (K-1)));
}

//Driver code
int main()
{
    int num = 10, K = 4;

    //Function call
    cout << LSB(num, K);

    return 0;
}
```

java

```
 // java code to print 'K'th LSB
import java .io.*;

class GFG {

    // Function returns 1 if set, 0 if not
    static boolean LSB(int num, int K)
    {
        boolean x = (num & (1 << (K-1))) != 0;
        return (x);
    }

    // Driver code
     public static void main(String[] args)
    {
        int num = 10, K = 4;

        //Function call
        if(LSB(num, K))
            System.out.println("1") ;

        else
            System.out.println("0");
    }
}

// This code is contributed by Anuj_67
```

Python

```
 # Python code to print 'K'th LSB

# Function returns 1 if set, 0 if not
def LSB(num, K):
    return bool(num & (1 << (K - 1) ))

# Driver code
num, k = 10, 4

res = LSB(num, k)
if res :
    print 1
else:
    print 0

#This code is contributed by Sachin Bisht
```

C#

```
 // C# code to print 'K'th LSB
using System;

class GFG {

    // Function returns 1 if set, 0 if not
    static bool LSB(int num, int K)
    {
        bool x = (num & (1 << (K-1))) != 0;
        return (x);
    }

    // Driver code
    static void Main()
    {
        int num = 10, K = 4;

        //Function call
        if(LSB(num, K))
            Console.Write("1") ;

        else
            Console.Write("0");
    }
}

// This code is contributed by Anuj_67
```

PHP

```
 <?php

// PHP code to print 'K'th LSB

// Function returns 1 if set, 0 if not
function LSB($num, $K)
{
    return ($num & (1 << ($K - 1)));
}

// Driver code
$num = 10;
$K = 4;

$r = LSB($num, $K);
if($r)
    echo '1';
else
    echo '0';

// This code is contributed by Ajit
?>
```

Output :

```
1
```

Improved By : jit_t, vt_m

Source

https://www.geeksforgeeks.org/print-kth-least-significant-bit-number/

Chapter 265

Program to count number of set bits in an (big) array

Program to count number of set bits in an (big) array - GeeksforGeeks

Given an integer array of length N (an arbitrarily large number). How to count number of set bits in the array?

The simple approach would be, create an efficient method to count set bits in a word (most prominent size, usually equal to bit length of processor), and add bits from individual elements of array.

Various methods of counting set bits of an integer exists, see this for example. These methods run at best O(logN) where N is number of bits. Note that on a processor N is fixed, count can be done in O(1) time on 32 bit machine irrespective of total set bits. Overall, the bits in array can be computed in O(n) time, where 'n' is array size.

However, a table look up will be more efficient method when array size is large. Storing table look up that can handle 2^{32} integers will be impractical.

The following code illustrates simple program to count set bits in a randomly generated 64 K integer array. The idea is to generate a look up for first 256 numbers (one byte), and break every element of array at byte boundary. A meta program using C/C++ preprocessor generates the look up table for counting set bits in a byte.

The mathematical derivation behind meta program is evident from the following table (Add the column and row indices to get the number, then look into the table to get set bits in that number. For example, to get set bits in 10, it can be extracted from row named as 8 and column named as 2),

```
   0, 1, 2, 3
 0 - 0, 1, 1, 2 -------- GROUP_A(0)
 4 - 1, 2, 2, 3 -------- GROUP_A(1)
 8 - 1, 2, 2, 3 -------- GROUP_A(1)
12 - 2, 3, 3, 4 -------- GROUP_A(2)
```

```
16 - 1, 2, 2, 3 -------- GROUP_A(1)
20 - 2, 3, 3, 4 -------- GROUP_A(2)
24 - 2, 3, 3, 4 -------- GROUP_A(2)
28 - 3, 4, 4, 5 -------- GROUP_A(3) ... so on
```

From the table, there is a patten emerging in multiples of 4, both in the table as well as in the group parameter. The sequence can be generalized as shown in the code.

Complexity:

All the operations takes O(1) except iterating over the array. The time complexity is O(n) where 'n' is size of array. Space complexity depends on the meta program that generates look up.

Code:

```
 #include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* Size of array 64 K */
#define SIZE (1 << 16)

/* Meta program that generates set bit count
   array of first 256 integers */

/* GROUP_A - When combined with META_LOOK_UP
   generates count for 4x4 elements */

#define GROUP_A(x) x, x + 1, x + 1, x + 2

/* GROUP_B - When combined with META_LOOK_UP
   generates count for 4x4x4 elements */

#define GROUP_B(x) GROUP_A(x), GROUP_A(x+1), GROUP_A(x+1), GROUP_A(x+2)

/* GROUP_C - When combined with META_LOOK_UP
   generates count for 4x4x4x4 elements */

#define GROUP_C(x) GROUP_B(x), GROUP_B(x+1), GROUP_B(x+1), GROUP_B(x+2)

/* Provide appropriate letter to generate the table */

#define META_LOOK_UP(PARAMETER) \
   GROUP_##PARAMETER(0),  \
   GROUP_##PARAMETER(1),  \
   GROUP_##PARAMETER(1),  \
   GROUP_##PARAMETER(2)   \
```

```
int countSetBits(int array[], size_t array_size)
{
   int count = 0;

   /* META_LOOK_UP(C) - generates a table of 256 integers whose
      sequence will be number of bits in i-th position
      where 0 <= i < 256
   */

    /* A static table will be much faster to access */
       static unsigned char const look_up[] = { META_LOOK_UP(C) };

    /* No shifting funda (for better readability) */
    unsigned char *pData = NULL;

   for(size_t index = 0; index < array_size; index++)
   {
      /* It is fine, bypass the type system */
      pData = (unsigned char *)&array[index];

      /* Count set bits in individual bytes */
      count += look_up[pData[0]];
      count += look_up[pData[1]];
      count += look_up[pData[2]];
      count += look_up[pData[3]];
   }

   return count;
}

/* Driver program, generates table of random 64 K numbers */
int main()
{
   int index;
   int random[SIZE];

   /* Seed to the random-number generator */
   srand((unsigned)time(0));

   /* Generate random numbers. */
   for( index = 0; index < SIZE; index++ )
   {
      random[index] = rand();
   }

   printf("Total number of bits = %d\n", countSetBits(random, SIZE));
   return 0;
}
```

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

Source

https://www.geeksforgeeks.org/program-to-count-number-of-set-bits-in-an-big-array/

Chapter 266

Program to find parity

Program to find parity - GeeksforGeeks

Parity: Parity of a number refers to whether it contains an odd or even number of 1-bits. The number has "odd parity", if it contains odd number of 1-bits and is "even parity" if it contains even number of 1-bits.
Main idea of the below solution is – Loop while n is not 0 and in loop unset one of the set bits and invert parity.

```
Algorithm: getParity(n)
1. Initialize parity = 0
2. Loop while n != 0
      a. Invert parity
             parity = !parity
      b. Unset rightmost set bit
             n = n & (n-1)
3. return parity

Example:
 Initialize: n = 13 (1101)   parity = 0

n = 13 & 12  = 12 (1100)   parity = 1
n = 12 & 11 = 8  (1000)   parity = 0
n = 8 & 7 = 0  (0000)    parity = 1
```

Program:

C

```
 # include <stdio.h>
# define  bool int
```

```
/* Function to get parity of number n. It returns 1
   if n has odd parity, and returns 0 if n has even
   parity */
bool getParity(unsigned int n)
{
    bool parity = 0;
    while (n)
    {
        parity = !parity;
        n      = n & (n - 1);
    }
    return parity;
}

/* Driver program to test getParity() */
int main()
{
    unsigned int n = 7;
    printf("Parity of no %d = %s",  n,
             (getParity(n)? "odd": "even"));

    getchar();
    return 0;
}
```

Java

```
 // Java program to find parity
// of an integer
import java.util.*;
import java.lang.*;
import java.io.*;
import java.math.BigInteger;

class GFG
 {
    /* Function to get parity of number n.
    It returns 1 if n has odd parity, and
    returns 0 if n has even parity */
    static boolean getParity(int n)
    {
        boolean parity = false;
        while(n != 0)
        {
            parity = !parity;
            n = n & (n-1);
        }
```

```
        return parity;

    }

    /* Driver program to test getParity() */
    public static void main (String[] args)
    {
        int n = 12;
        System.out.println("Parity of no " + n + " = " +
                           (getParity(n)? "odd": "even"));
    }
}
/* This code is contributed by Amit khandelwal*/
```

Python3

```
 # Python3 code to get parity.

# Function to get parity of number n.
# It returns 1 if n has odd parity,
# and returns 0 if n has even parity
def getParity( n ):
    parity = 0
    while n:
        parity = ~parity
        n = n & (n - 1)
    return parity

# Driver program to test getParity()
n = 7
print ("Parity of no ", n," = ",
     ( "odd" if getParity(n) else "even"))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# program to find parity of an integer
using System;

class GFG {

    /* Function to get parity of number n.
    It returns 1 if n has odd parity, and
    returns 0 if n has even parity */
    static bool getParity(int n)
    {
```

```
        bool parity = false;
        while(n != 0)
        {
            parity = !parity;
            n = n & (n-1);
        }
        return parity;

    }

    // Driver code
    public static void Main ()
    {
        int n = 7;
        Console.Write("Parity of no " + n
                 + " = " + (getParity(n)?
                          "odd": "even"));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to find the parity
// of an unsigned integer

// Function to get parity of
// number n. It returns 1
// if n has odd parity, and
// returns 0 if n has even
// parity
function getParity( $n)
{
    $parity = 0;
    while ($n)
    {
        $parity = !$parity;
        $n = $n & ($n - 1);
    }
    return $parity;
}

    // Driver Code
    $n = 7;
    echo "Parity of no ",$n ," = " ,
          getParity($n)? "odd": "even";
```

```
// This code is contributed by anuj_67.
?>
```

Output:

```
Parity of no 7 = odd
```

Above solution can be optimized by using lookup table. Please refer to Bit Twiddle Hacks[1st reference] for details.

Time Complexity: The time taken by above algorithm is proportional to the number of bits set. Worst case complexity is O(Log n).

Uses: Parity is used in error detection and cryptography.

Compute the parity of a number using XOR and table look-up

References:
http://graphics.stanford.edu/~seander/bithacks.html#ParityNaive – last checked on 30 May 2009.

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/program-to-find-parity/

Chapter 267

Program to find whether a no is power of two

Program to find whether a no is power of two - GeeksforGeeks

Given a positive integer, write a function to find if it is a power of two or not.

Examples :

```
Input : n = 4
Output : Yes
22 = 4

Input : n = 7
Output : No

Input : n = 32
Output : Yes
25 = 32
```

1. A simple method for this is to simply take the log of the number on base 2 and if you get an integer then number is power of 2.

C

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

/* Function to check if x is power of 2*/
bool isPowerOfTwo(int n)
{
```

```
   return (ceil(log2(n)) == floor(log2(n)));
}

// Driver program
int main()
{
    isPowerOfTwo(31)? printf("Yes\n"): printf("No\n");
    isPowerOfTwo(64)? printf("Yes\n"): printf("No\n");
    return 0;
}

// This code is contributed by bibhudhendra
```

PHP

```
 <?php
// PHP Program to find
// whether a no is
// power of two

// Function to check
// Log base 2
function Log2($x)
{
    return (log10($x) /
            log10(2));
}


// Function to check
// if x is power of 2
function isPowerOfTwo($n)
{
    return (ceil(Log2($n)) ==
            floor(Log2($n)));
}

// Driver Code
if(isPowerOfTwo(31))
echo "Yes\n";
else
echo "No\n";

if(isPowerOfTwo(64))
echo "Yes\n";
else
echo "No\n";
```

```
// This code is contributed
// by Sam007
?>
```

Python3

```
# Python3 Program to find
# whether a no is
# power of two
import math

# Function to check
# Log base 2
def Log2(x):
return (math.log10(x) /
math.log10(2));

# Function to check
# if x is power of 2
def isPowerOfTwo(n):
return (math.ceil(Log2(n)) ==
math.floor(Log2(n)));

# Driver Code
if(isPowerOfTwo(31)):
print("Yes");
else:
print("No");

if(isPowerOfTwo(64)):
print("Yes");
else:
print("No");

# This code is contributed
# by mits
```

Output:

```
No
Yes
```

2. Another solution is to keep dividing the number by two, i.e, do n = n/2 iteratively. In any iteration, if n%2 becomes non-zero and n is not 1 then n is not a power of 2. If n becomes 1 then it is a power of 2.

C

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

```
/* Function to check if x is power of 2*/
bool isPowerOfTwo(int n)
{
  if (n == 0)
    return 0;
  while (n != 1)
  {
      if (n%2 != 0)
         return 0;
      n = n/2;
  }
  return 1;
}

/*Driver program to test above function*/
int main()
{
  isPowerOfTwo(31)? printf("Yes\n"): printf("No\n");
  isPowerOfTwo(64)? printf("Yes\n"): printf("No\n");
  return 0;
}
```

Java

```
 // Java program to find whether
// a no is power of two
import java.io.*;

class GFG {

    // Function to check if
    // x is power of 2
    static boolean isPowerOfTwo(int n)
    {
        if (n == 0)
            return false;

        while (n != 1)
        {
            if (n % 2 != 0)
                return false;
            n = n / 2;
        }
        return true;
    }

    // Driver program
```

```
    public static void main(String args[])
    {
        if (isPowerOfTwo(31))
            System.out.println("Yes");
        else
            System.out.println("No");

        if (isPowerOfTwo(64))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

// This code is contributed by Nikita tiwari.
```

Python3

```
 # Python program to check if given
# number is power of 2 or not

# Function to check if x is power of 2
def isPowerOfTwo(n):
    if (n == 0):
        return False
    while (n != 1):
            if (n % 2 != 0):
                return False
            n = n // 2

    return True

# Driver code
if(isPowerOfTwo(31)):
    print('Yes')
else:
    print('No')
if(isPowerOfTwo(64)):
    print('Yes')
else:
    print('No')

# This code is contributed by Danish Raza
```

C#

```
 // C# program to find whether
```

```
// a no is power of two
using System;

class GFG
{

    // Function to check if
    // x is power of 2
    static bool isPowerOfTwo(int n)
    {
        if (n == 0)
            return false;

        while (n != 1) {
            if (n % 2 != 0)
                return false;

            n = n / 2;
        }
        return true;
    }

    // Driver program
    public static void Main()
    {
        Console.WriteLine(isPowerOfTwo(31) ? "Yes" : "No");
        Console.WriteLine(isPowerOfTwo(64) ? "Yes" : "No");

    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php

// Function to check if
// x is power of 2
function isPowerOfTwo($n)
{
if ($n == 0)
    return 0;
while ($n != 1)
{
    if ($n % 2 != 0)
        return 0;
    $n = $n / 2;
```

```
}
return 1;
}

// Driver Code
if(isPowerOfTwo(31))
    echo "Yes\n";
else
    echo "No\n";

if(isPowerOfTwo(64))
    echo "Yes\n";
else
    echo "No\n";

// This code is contributed
// by Sam007
?>
```

Output :

```
No
Yes
```

3. All power of two numbers have only one bit set. So count the no. of set bits and if you get 1 then number is a power of 2. Please see Count set bits in an integer for counting set bits.

4. If we subtract a power of 2 numbers by 1 then all unset bits after the only set bit become set; and the set bit become unset.

For example for 4 (100) and 16(10000), we get following after subtracting 1
3 –> 011
15 –> 01111

So, if a number n is a power of 2 then bitwise & of n and n-1 will be zero. We can say n is a power of 2 or not based on value of n&(n-1). The expression n&(n-1) will not work when n is 0. To handle this case also, our expression will become n& (!n&(n-1)) (thanks to https://www.geeksforgeeks.org/program-to-find-whether-a-no-is-power-of-two/Mohammad for adding this case).
Below is the implementation of this method.

C

```
 #include<stdio.h>
#define bool int

/* Function to check if x is power of 2*/
```

```
bool isPowerOfTwo (int x)
{
  /* First x in the below expression is for the case when x is 0 */
  return x && (!(x&(x-1)));
}

/*Driver program to test above function*/
int main()
{
  isPowerOfTwo(31)? printf("Yes\n"): printf("No\n");
  isPowerOfTwo(64)? printf("Yes\n"): printf("No\n");
  return 0;
}
```

Java

```
 // Java program to efficiently
// check for power for 2

class Test
{
    /* Method to check if x is power of 2*/
    static boolean isPowerOfTwo (int x)
    {
      /* First x in the below expression is
        for the case when x is 0 */
        return x!=0 && ((x&(x-1)) == 0);

    }

    // Driver method
    public static void main(String[] args)
    {
         System.out.println(isPowerOfTwo(31) ? "Yes" : "No");
         System.out.println(isPowerOfTwo(64) ? "Yes" : "No");

    }
}
// This program is contributed by Gaurav Miglani
```

Python

```
 # Python program to check if given
# number is power of 2 or not

# Function to check if x is power of 2
def isPowerOfTwo (x):
```

```
    # First x in the below expression
    # is for the case when x is 0
    return (x and (not(x & (x - 1))) )

# Driver code
if(isPowerOfTwo(31)):
    print('Yes')
else:
    print('No')

if(isPowerOfTwo(64)):
    print('Yes')
else:
    print('No')

# This code is contributed by Danish Raza
```

C#

```
 // C# program to efficiently
// check for power for 2
using System;

class GFG
{
    // Method to check if x is power of 2
    static bool isPowerOfTwo (int x)
    {
        // First x in the below expression
        // is for the case when x is 0
        return x != 0 && ((x & (x - 1)) == 0);

    }

    // Driver method
    public static void Main()
    {
        Console.WriteLine(isPowerOfTwo(31) ? "Yes" : "No");
        Console.WriteLine(isPowerOfTwo(64) ? "Yes" : "No");

    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to efficiently
// check for power for 2

// Function to check if
// x is power of 2
function isPowerOfTwo ($x)
{
// First x in the below expression
// is for the case when x is 0
return $x && (!($x & ($x - 1)));
}

// Driver Code
if(isPowerOfTwo(31))
    echo "Yes\n" ;
else
    echo "No\n";

if(isPowerOfTwo(64))
    echo "Yes\n" ;
else
    echo "No\n";

// This code is contributed by Sam007
?>
```

Output :

```
No
Yes
```

Improved By : Sam007, Mithun Kumar

Source

https://www.geeksforgeeks.org/program-to-find-whether-a-no-is-power-of-two/

Chapter 268

Program to invert bits of a number Efficiently

Program to invert bits of a number Efficiently - GeeksforGeeks

Given a non-negative integer N. The task is to invert the bits of the number N and print the decimal equivalent of the number obtained after inverting the bits.

Note: Leading 0's are not being considered.

Examples:

```
Input : 11
Output : 4
(11)10 = (1011)2
After inverting the bits, we get:
(0100)2 = (4)10.

Input : 20
Output : 11
(20)10 = (10100)2.
After inverting the bits, we get:
(01011)2 = (11)10.
```

The similar problem is already discussed in Invert actual bits of a number.

In this article, an efficient approach using bitwise operators is discussed. Below is the step by step algorithm to solve the problem:

1. Calculate the total number of bits in the given number. This can be done by calculating:

```
X = log2N
```

Where, N is the given number and X is the total number of bits of N.

2. The next step is to generate a number with X bits and all bits set. That is, *11111….X-times*. This can be done by calculating:

```
Step-1: M = 1 << X
Step-2: M = M | (M-1)
```

Where M is the required X-bit number with all bits set.

3. The final step is to calculate the bit-wise XOR of M with N, which will be our answer.

Below is the implementation of the above approach:

```
 // CPP program to invert actual bits
// of a number.
#include <bits/stdc++.h>

using namespace std;

// Function to invert bits of a number
int invertBits(int n)
{
    // Calculate number of bits of N-1;
    int x = log2(n) ;

    int m = 1 << x;

    m = m | m - 1;

    n = n ^ m;

    return n;
}

// Driver code
int main()
{
    int n = 20;

    cout << invertBits(n);

    return 0;
}
```

Output:

```
11
```

Time Complexity: O($\log_2$n)
Auxiliary Space: O(1)

Improved By : ayushjauhari14

Source

https://www.geeksforgeeks.org/program-to-invert-bits-of-a-number-efficiently/

Chapter 269

Python Slicing | Extract 'k' bits from a given position

Python Slicing | Extract 'k' bits from a given position - GeeksforGeeks

How to extract 'k' bits from a given position 'p' in a number?

Examples:

```
Input : number = 171
             k = 5
             p = 2
Output : The extracted number is 21
171 is represented as 10101011 in binary,
so, you should get only 10101 i.e. 21.

Input : number = 72
            k = 5
            p = 1
Output : The extracted number is 8
72 is represented as 1001000 in binary,
so, you should get only 01000 i.e 8.
```

We have existing solution for this problem please refer Extract 'k' bits from a given position in a number link. We can solve this problem quickly in python using slicing. Approach is simple,

1. Convert given number into it's binary using bin() function and remove first two characters '0b' from it, because bin function appends '0b' as prefix in output binary string.
2. We need to start extracting k bits from starting position p from right, that means end index of extracted sub-string will be **end = (len(binary) − p)** and start index will be **start = end − k + 1** of original binary string.

3. Convert extracted sub-string into decimal again.

```
 # Function to extract 'k' bits from a given
# position in a number

def extractKBits(num,k,p):

     # convert number into binary first
     binary = bin(num)

     # remove first two characters
     binary = binary[2:]

     end = len(binary) - p
     start = end - k + 1

     # extract k  bit sub-string
     kBitSubStr = binary[start : end+1]

     # convert extracted sub-string into decimal again
     print (int(kBitSubStr,2))

# Driver program
if __name__ == "__main__":
    num = 171
    k = 5
    p = 2
    extractKBits(num,k,p)
```

Output:

```
21
```

Source

https://www.geeksforgeeks.org/python-slicing-extract-k-bits-given-position/

Chapter 270

Python map function | Count total set bits in all numbers from 1 to n

Python map function | Count total set bits in all numbers from 1 to n - GeeksforGeeks

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
Binary representations are 1, 2 and 3
1, 10 and 11 respectively. Total set
bits are 1 + 1 + 2 = 4.

Input: n = 6
Output: 9

Input: n = 7
Output: 12

Input: n = 8
Output: 13
```

We have existing solution for this problem please refer Count total set bits in all numbers from 1 to n link. We can solve this problem in python using map() function. Approach is very simple,

1. Write a function which first converts number into binary using bin(num) function and returns count of set bits in it.
2. Map user defined function on list of numbers from 1 to n and we will get list of individual count of set bits in each number.
3. Sum up count of all set bits.

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

# user defined function
def countSetBit(num):

     # convert decimal value into binary and
     # count all 1's in it
     binary = bin(num)

     return len([ch for ch in binary if ch=='1'])

# function which count set bits in each number
def countSetBitAll(input):

    # map count function on each number
    print (sum(map(countSetBit,input)))

# Driver program
if __name__ == "__main__":
    n = 8
    input=[]
    for i in range(1,n+1):
         input.append(i)
    countSetBitAll(input)
```

Output:

```
13
```

Source

https://www.geeksforgeeks.org/python-map-function-count-total-set-bits-numbers-1-n/

Chapter 271

Python program to convert floating to binary

Python program to convert floating to binary - GeeksforGeeks

Python doesn't provide any inbuilt method to easily convert floating point decimal numbers to binary number. So, Let's do this manually.

Approach :
To convert a floating point decimal number into binary, first convert the integer part into binary form and then fractional part into binary form and finally combine both results to get the final answer.
For Integer Part, keep dividing the number by 2 and noting down the remainder until and unless the dividend is less than 2. If so, stop and copy all the remainders together.
For Decimal Part, keep multiplying the decimal part with 2 until and unless 0 left as fractional part. After multiplying the first time, note down integral part and again multiply decimal part of the new value by 2. Keep doing this until reached a perfect number.

> **Above steps can be written as :**
> 1(base 10) = 1(base 2) and .234(base 10) = .0011(base 2)
>
> Now, to get the binary of 1.234, merge both results as a complete number.
>
> `(1)10 = (1)2 (.234)10 = (.0011)2 (1.234)10 = (1.0011...)2 (1.234)10 = (1.0011)2` [approx.]

Below is the implementation :

```
 # Python program to convert float
# decimal to binary number

# Function returns octal representation
def float_bin(number, places = 3):
```

```
    # split() seperates whole number and decimal
    # part and stores it in two seperate variables
    whole, dec = str(number).split(".")

    # Convert both whole number and decimal
    # part from string type to integer type
    whole = int(whole)
    dec = int (dec)

    # Convert the whole number part to it's
    # respective binary form and remove the
    # "0b" from it.
    res = bin(whole).lstrip("0b") + "."

    # Iterate the number of times, we want
    # the number of decimal places to be
    for x in range(places):

        # Multiply the decimal value by 2
        # and seperate the whole number part
        # and decimal part
        whole, dec = str((decimal_converter(dec)) * 2).split(".")

        # Convert the decimal part
        # to integer again
        dec = int(dec)

        # Keep adding the integer parts
        # receive to the result variable
        res += whole

    return res

# Function converts the value passed as
# parameter to it's decimal representation
def decimal_converter(num):
    while num > 1:
        num /= 10
    return num

# Driver Code

# Take the user input for
# the floating point number
n = input("Enter your floating point value : \n")

# Take user input for the number of
```

```
# decimal places user want result as
p = int(input("Enter the number of decimal places of the result : \n"))

print(float_bin(n, places = p))
```

Output :

```
Enter your floating point value :
1.234
Enter the number of decimal places of the result :
4

1.0011


Enter your floating point value :
11.234
Enter the number of decimal places of the result :
4

1011.0011
```

Source

https://www.geeksforgeeks.org/python-program-to-convert-floating-to-binary/

Chapter 272

Python | Count set bits in a range

Python | Count set bits in a range - GeeksforGeeks

Given a non-negative number **n** and two values **l** and **r**. The problem is to count the number of set bits in the range **l** to **r** in the binary representation of **n**, i.e, to count set bits from the rightmost **lth** bit to the rightmost **rth** bit.

Constraint: 1 <= l <= r <= number of bits in the binary representation of **n**.

Examples:

```
Input : n = 42, l = 2, r = 5
Output : 2
(42)10 = (101010)2
There are '2' set bits in the range 2 to 5.

Input : n = 79, l = 1, r = 4
Output : 4
```

We have existing solution for this problem please refer Count set bits in a range link. We can solve this problem quickly in Python. Approach is very simple,

1. Convert decimal into binary using bin(num) function.
2. Now remove first two characters of output binary string because bin function appends '0b' as prefix in output string by default.
3. Slice string starting from index **(l-1)** to index **r** and reverse it, then count set bits in between.

```
# Function to count set bits in a range
```

```
def countSetBits(n,l,r):

    # convert n into it's binary
    binary = bin(n)

    # remove first two characters
    binary = binary[2:]

    # reverse string
    binary = binary[-1::-1]

    # count all set bit '1' starting from index l-1
    # to r, where r is exclusive
    print (len([binary[i] for i in range(l-1,r) if binary[i]=='1']))

# Driver program
if __name__ == "__main__":
    n=42
    l=2
    r=5
    countSetBits(n,l,r)
```

Output:

```
2
```

Source

https://www.geeksforgeeks.org/python-count-set-bits-range/

Chapter 273

Python | Count unset bits in a range

Python | Count unset bits in a range - GeeksforGeeks

Given a non-negative number n and two values l and r. The problem is to count the number of unset bits in the range l to r in the binary representation of n, i.e, to count unset bits from the rightmost lth bit to the rightmost rth bit.

Examples:

```
Input : n = 42, l = 2, r = 5
Output : 2
(42)10 = (101010)2
There are '2' unset bits in the range 2 to 5.

Input : n = 80, l = 1, r = 4
Output : 4
```

We have existing solution for this problem please refer Count unset bits in a range link. We can solve this problem quickly in Python. Approach is very simple,

1. Convert decimal into binary using bin(num) function.
2. Now remove first two characters of output binary string because bin function appends '0b' as prefix in output string by default.
3. Slice string starting from index **(l-1)** to index **r** and reverse it, then count unset bits in between.

```
 # Function to count unset bits in a range

def unsetBits(n,l,r):
```

```
    # convert n into it's binary
    binary = bin(n)

    # remove first two characters
    binary = binary[2:]

    # reverse string
    binary = binary[-1::-1]

    # count all unset bit '0' starting from index l-1
    # to r, where r is exclusive
    print (len([binary[i] for i in range(l-1,r) if binary[i]=='0']))

# Driver program
if __name__ == "__main__":
    n=42
    l=2
    r=5
    unsetBits(n,l,r)
```

Output:

```
2
```

Source

https://www.geeksforgeeks.org/python-count-unset-bits-range/

Chapter 274

Queries for number of array elements in a range with Kth Bit Set

Queries for number of array elements in a range with Kth Bit Set - GeeksforGeeks

Given an array of **N** positive (32-bit)integers, the task is to answer **Q** queries of the following form:

```
Query(L, R, K): Print the number of elements of the array in the
                range L to R, which have their Kth bit as set
```

Note: Consider **LSB** to be indexed at **1**.

Examples:

> **Input** : arr[] = { 8, 9, 1, 3 }
> Query 1: L = 1, R = 3, K = 4
> Query 2: L = 2, R = 4, K = 1
> **Output** :
> 2
> 3
>
> **Explanation**:
> For the 1st query, the range (1, 3) represents elements, {8, 9, 1}. Among these elements only **8 and 9** have their 4th bit set. Thus, the answer for this query is **2**.
>
> For the 2nd query, the range (2, 4) represents elements, {9, 1, 3}. **All** of these elements have their 1st bit set. Thus, the answer for this query is **3**.

Prerequisites: Bit Manipulation | Prefix Sum Arrays

Method 1 (Brute Force) : For each query, traverse the array from L to R, and at every index check if the array element at that index has its K^{th} bit as set. If it does increment the counter variable.

Below is the implementation of above approach.

C++

```
 /* C++ Program to find the number of elements
   in a range L to R having the Kth bit as set */
#include <bits/stdc++.h>
using namespace std;

// Maximum bits required in binary represention
// of an array element
#define MAX_BITS 32

/* Returns true if n has its kth bit as set,
   else returns false */
bool isKthBitSet(int n, int k)
{
    if (n & (1 << (k - 1)))
        return true;
    return false;
}

/* Returns the answer for each query with range L
   to R querying for the number of elements with
   the Kth bit set in the range */
int answerQuery(int L, int R, int K, int arr[])
{
    // counter stores the number of element in
    // the range with the kth bit set
    int counter = 0;
    for (int i = L; i <= R; i++) {
        if (isKthBitSet(arr[i], K)) {
            counter++;
        }
    }
    return counter;
}

// Print the answer for all queries
void answerQueries(int queries[][3], int Q,
                   int arr[], int N)
```

```
{
    int query_L, query_R, query_K;

    for (int i = 0; i < Q; i++) {
        query_L = queries[i][0] - 1;
        query_R = queries[i][1] - 1;
        query_K = queries[i][2];

        cout << "Result for Query " << i + 1 << " = "
             << answerQuery(query_L, query_R, query_K, arr)
             << endl;
    }
}

// Driver Code
int main()
{

    int arr[] = { 8, 9, 1, 3 };
    int N = sizeof(arr) / sizeof(arr[0]);

    /* queries[][] denotes the array of queries
    where each query has three integers
    query[i][0] -> Value of L for ith query
    query[i][0] -> Value of R for ith query
    query[i][0] -> Value of K for ith query */
    int queries[][3] = {
        { 1, 3, 4 },
        { 2, 4, 1 }
    };
    int Q = sizeof(queries) / sizeof(queries[0]);

    answerQueries(queries, Q, arr, N);

    return 0;
}
```

Java

```
 // Java Program to find the
// number of elements in a
// range L to R having the
// Kth bit as set
import java.util.*;
import java.lang.*;
import java.io.*;

// Maximum bits required
```

```
// in binary represention
// of an array element
class GFG
{
static final int MAX_BITS = 32;

/* Returns true if n
has its kth bit as set,
else returns false */
static boolean isKthBitSet(int n,
                           int k)
{
    if ((n & (1 << (k - 1))) != 0)
        return true;
    return false;
}

/* Returns the answer for
each query with range L
to R querying for the number
of elements with the Kth bit
set in the range */
static int answerQuery(int L, int R,
                       int K, int arr[])
{
    // counter stores the number
    // of element in the range
    // with the kth bit set
    int counter = 0;
    for (int i = L; i <= R; i++)
    {
        if (isKthBitSet(arr[i], K))
        {
            counter++;
        }
    }
    return counter;
}

// Print the answer
// for all queries
static void answerQueries(int queries[][], int Q,
                          int arr[], int N)
{
    int query_L, query_R, query_K;

    for (int i = 0; i < Q; i++)
    {
```

```
        query_L = queries[i][0] - 1;
        query_R = queries[i][1] - 1;
        query_K = queries[i][2];

        System.out.println("Result for Query " +
                           (i + 1) + " = " +
                answerQuery(query_L, query_R,
                           query_K, arr));

    }
}

// Driver Code
public static void main(String args[])
{
    int arr[] = { 8, 9, 1, 3 };
    int N = arr.length;

    /* queries[][] denotes the array
    of queries where each query has
    three integers
    query[i][0] -> Value of L for ith query
    query[i][0] -> Value of R for ith query
    query[i][0] -> Value of K for ith query */
    int queries[][] =
    {
        { 1, 3, 4 },
        { 2, 4, 1 }
    };
    int Q = queries.length;

    answerQueries(queries, Q, arr, N);
}
}

// This code is contributed
// by Subhadeep
```

Output:

```
Result for Query 1 = 2
Result for Query 2 = 3
```

Time Complexity : O(N) for each query.

Method 2 (Efficient) : Assuming that every integer in the array has at max **32** bits in its Binary Representation. A 2D prefix sum array can be built to solve the problem. Here

the 2^{nd} dimension of the prefix array is of size equal to the **maximum number of bits** required to represent a integer of the array in binary.

Let the Prefix Sum Array be **P**[][]. Now, **P**[**i**][**j**] denotes the number of Elements from **0** to **i**, which have their $\mathbf{j^{th}}$ bit as set. This prefix sum array is built before answering the queries. If a query from L to R is encountered, querying for elements in this range having their K^{th} bit as set, then the answer for that query is $\mathbf{P[R][K] - P[L - 1][K]}$.

Below is the implementation of above approach.

C++

```
 /* C++ Program to find the number of elements
   in a range L to R having the Kth bit as set */
#include <bits/stdc++.h>
using namespace std;

// Maximum bits required in binary represention
// of an array element
#define MAX_BITS 32

/* Returns true if n has its kth bit as set,
   else returns false */
bool isKthBitSet(int n, int k)
{
    if (n & (1 << (k - 1)))
        return true;
    return false;
}

// Return pointer to the prefix sum array
int** buildPrefixArray(int N, int arr[])
{
    // Build a prefix sum array P[][]
    // where P[i][j] represents the number of
    // elements from 0 to i having the jth bit as set
    int** P = new int*[N + 1];
    for (int i = 0; i <= N; ++i) {
        P[i] = new int[MAX_BITS + 1];
    }

    for (int i = 0; i <= MAX_BITS; i++) {
        P[0][i] = 0;
    }

    for (int i = 0; i < N; i++) {

        for (int j = 1; j <= MAX_BITS; j++) {
```

```
            // prefix sum from 0 to i for each bit
            // position jhas the value of sum from 0
            // to i-1 for each j
            if (i)
                P[i][j] = P[i - 1][j];

            // if jth bit set then increment P[i][j] by 1
            bool isJthBitSet = isKthBitSet(arr[i], j);
            if (isJthBitSet) {
                P[i][j]++;
            }
        }
    }

    return P;
}

/* Returns the answer for each query with range
   L to R querying for the number of elements with
   the Kth bit set in the range */
int answerQuery(int L, int R, int K, int** P)
{

    /* Number of elements in range L to R with Kth
       bit set = (Number of elements from 0 to R with
       kth bit set) - (Number of elements from 0 to L-1
       with kth bit set) */
    if (L)
        return P[R][K] - P[L - 1][K];
    else
        return P[R][K];
}

// Print the answer for all queries
void answerQueries(int queries[][3], int Q,
                   int arr[], int N)
{

    // Build Prefix Array to answer queries efficiently
    int** P = buildPrefixArray(N, arr);

    int query_L, query_R, query_K;

    for (int i = 0; i < Q; i++) {
        query_L = queries[i][0] - 1;
        query_R = queries[i][1] - 1;
        query_K = queries[i][2];
```

```
        cout << "Result for Query " << i + 1 << " = "
             << answerQuery(query_L, query_R, query_K, P)
             << endl;
    }
}

// Driver Code
int main()
{

    int arr[] = { 8, 9, 1, 3 };
    int N = sizeof(arr) / sizeof(arr[0]);

    /* queries[][] denotes the array of queries
    where each query has three integers
    query[i][0] -> Value of L for ith query
    query[i][0] -> Value of R for ith query
    query[i][0] -> Value of K for ith query */
    int queries[][3] = {
        { 1, 3, 4 },
        { 2, 4, 1 }
    };
    int Q = sizeof(queries) / sizeof(queries[0]);

    answerQueries(queries, Q, arr, N);

    return 0;
}
```

Output:

```
Result for Query 1 = 2
Result for Query 2 = 3
```

Time Complexity of building the Prefix array is O(N * Maximum number of Bits) and each query is answered in O(1).
Auxiliary space : O(N * Maximum Number of Bits) is required to build the Prefix Sum Array

Improved By : tufan_gupta2000

Source

https://www.geeksforgeeks.org/queries-for-number-of-array-elements-in-a-range-with-kth-bit-set/

Chapter 275

Queries on XOR of XORs of all subarrays

Queries on XOR of XORs of all subarrays - GeeksforGeeks

Given an array **A** of **n** integers, say A_1, A_2, A_3, ..., A_n. You are given **Q** queries of the form **[l, r]**. The task is to find the XOR of XORs of all the subarrays of an array having elements A_l, A_{l+1}, ..., A_r.

Examples:

```
Input : A[] = { 1, 2, 3, 4, 5 }, Q = 3
        q1 = { 1, 2 }
        q2 = { 1, 3 }
        q3 = { 2, 4 }
Output : 0
         2
         6
For query 1, the extracted array is [1, 2] and
subarrays of the array is [1], [2], [1, 2].
So, the answer is (1)   (2)   (1  2) = 0.
For query 2, the extracted array is [1, 2, 3] and
subarrays of the array is
[1], [2], [1, 2], [2, 3], [1, 2, 3].
So the answer is (1)   (2)   (3)  (1   2)
                 (2   3)   (1   2   3) = 2.
For query 3, the extracted array is [2, 3, 4] and
subarrays of the array is
[2], [3], [4], [2, 3], [3, 4], [2, 3, 4].
So the answer is (2)  (3)   (4)   (2   3)
                 (3   4)   (2   3   4) = 6.
```

```
Input : A[] = { 5, 8, 9, 1, 7 }, Q = 3
        query1 = { 1, 3 }
        query2 = { 3, 4 }
        query3 = { 2, 5 }
Output : 12
         0
         0
```

First of all recall the properties of XOR,
1. x x = 0
2. If x y = z, then x = y z
Using the first property, we can say that any number x XORed even number of times will result in a 0 and odd number of times will result in x.
If we want to find XOPR of XORs all the subarrays of an array, we need to find the elements which appear odd number of times in all subarrays in total.

Let's say we have an array [1, 2, 3]. Its subarrays will be [1], [2], [3], [1, 2], [2, 3], [1, 2, 3].
1 has occurred three times in total.
2 has occurred four times in total.
3 has occurred three times in total.

We can observe that number at i^{th} index will have (i + 1)x(sizeofarray – i) frequency.
If an array has odd number of integers, starting from the first element, every alternate element will appear odd number of times in all subarrays in total. Therefore, the XOR of XORs of all subarrays will be XOR of the alternate elements in the arrays.
If an array has even number of integers, every element will appear even number of times in all subarrays in total. Therefore, the XOR of XORs of all subarrays will always be 0.

Traversing the array for every query is inefficient. We can store value of XORs upto every elements by XORing it with the alternate elements using recurrence
prefix_xor[i] = A[i] prefix_xor[i – 2]

For every query, we have starting index as l and ending index as r. If (r – l + 1) odd, answer will be prefix_xor[r] prefix_xor[l – 2].

Below is the implementation of this approach:

C++

```
 // CPP Program to answer queries on XOR of XORs
// of all subarray
#include <bits/stdc++.h>
#define N 100
using namespace std;

// Output for each query
void ansQueries(int prefeven[], int prefodd[],
                                  int l, int r)
{
    // If number of element is even.
    if ((r - l + 1) % 2 == 0)
```

```
        cout << "0";

    // If number of element is odd.
    else {

        // if l is even
        if (l % 2 == 0)
            cout << (prefeven[r] ^ prefeven[l - 1]);

        // if l is odd
        else
            cout << (prefodd[r] ^ prefodd[l - 1]);
    }

    cout << endl;
}

// Wrapper Function
void wrapper(int arr[], int n, int l[], int r[], int q)
{
    int prefodd[N] = { 0 }, prefeven[N] = { 0 };

    // Evaluating prefixodd and prefixeven
    for (int i = 1; i <= n; i++) {
        if ((i) % 2 == 0) {
            prefeven[i] = arr[i - 1] ^ prefeven[i - 1];
            prefodd[i] = prefodd[i - 1];
        }
        else {
            prefeven[i] = prefeven[i - 1];
            prefodd[i] = prefodd[i - 1] ^ arr[i - 1];
        }
    }

    int i = 0;
    while (i != q) {
        query(prefeven, prefodd, l[i], r[i]);
        i++;
    }
}

// Driven Program
int main()
{
    int arr[] = { 1, 2, 3, 4, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);

    int l[] = { 1, 1, 2 };
```

```
    int r[] = { 2, 3, 4 };
    int q = sizeof(l) / sizeof(l[0]);

    ansQueries(arr, n, l, r, q);
    return 0;
}
```

Java

```
 // JAVA Code for Queries on XOR
// of XORs of all subarrays
import java.util.*;

class GFG {

    // Output for each query
    static void ansQueries(int prefeven[],
                           int prefodd[],
                           int l, int r)
    {
        // If number of element is even.
        if ((r - l + 1) % 2 == 0)
            System.out.println("0");

        // If number of element is odd.
        else
        {
            // if l is even
            if (l % 2 == 0)
                System.out.println(prefeven[r] ^
                                   prefeven[l - 1]);

            // if l is odd
            else
                System.out.println(prefodd[r] ^
                                    prefodd[l - 1]);
        }
    }

    // Wrapper Function
    static void wrapper(int arr[], int n,
                        int l[], int r[],
                                  int q)
    {
        int prefodd[] = new int[100];
        int prefeven[] = new int[100];

        // Evaluating prefixodd
```

```
        // and prefixeven
        for (int i = 1; i <= n; i++) {

            if ((i) % 2 == 0) {

                prefeven[i] = arr[i - 1] ^
                              prefeven[i - 1];

                prefodd[i] = prefodd[i - 1];
            }
            else
            {
                prefeven[i] = prefeven[i - 1];
                prefodd[i] = prefodd[i - 1] ^
                              arr[i - 1];
            }
        }

        int i = 0;

        while (i != q){

            ansQueries(prefeven, prefodd,
                              l[i], r[i]);
            i++;
        }
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int arr[] = {1, 2, 3, 4 , 5};
        int n = arr.length;

        int l[] = {1, 1, 2};
        int r[] = {2, 3, 4};
        int q = l.length;

        wrapper(arr, n, l, r, q);
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

C#

```
 // C# code for Queries on XOR
// of XORs of all subarrays
```

```
using System;

class GFG {

    // Output for each query
    static void ansQueries(int[] prefeven,
                           int[] prefodd,
                           int l, int r)
    {
        // If number of element is even.
        if ((r - l + 1) % 2 == 0)
            Console.WriteLine("0");

        // If number of element is odd.
        else {
            // if l is even
            if (l % 2 == 0)
                Console.WriteLine(prefeven[r] ^ prefeven[l - 1]);

            // if l is odd
            else
                Console.WriteLine(prefodd[r] ^ prefodd[l - 1]);
        }
    }

    // Wrapper Function
    static void wrapper(int[] arr, int n,
                        int[] l, int[] r,
                        int q)
    {
        int[] prefodd = new int[100];
        int[] prefeven = new int[100];

        // Evaluating prefixodd
        // and prefixeven
        for (int i = 1; i <= n; i++) {

            if ((i) % 2 == 0) {

                prefeven[i] = arr[i - 1] ^ prefeven[i - 1];

                prefodd[i] = prefodd[i - 1];
            }
            else {
                prefeven[i] = prefeven[i - 1];
                prefodd[i] = prefodd[i - 1] ^ arr[i - 1];
            }
        }
```

```
        int j = 0;

        while (j != q) {

            ansQueries(prefeven, prefodd,
                    l[j], r[j]);
            j++;
        }
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int[] arr = { 1, 2, 3, 4, 5 };
        int n = arr.Length;

        int[] l = { 1, 1, 2 };
        int[] r = { 2, 3, 4 };
        int q = l.Length;

        wrapper(arr, n, l, r, q);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// php Program to answer
// queries on XOR of XORs
// of all subarray
// Output for each query

function ansQueries($prefeven, $prefodd,
                                $l, $r)
{

    // If number of element is even.
    if (($r - $l + 1) % 2 == 0)
    {
        echo "0";
}

    // If number of element is odd.
    else {
```

```
        // if l is even
        if ($l % 2 == 0)
            echo ($prefeven[$r] ^
                  $prefeven[$l - 1]);

        // if l is odd
        else
            echo ($prefodd[$r] ^
                  $prefodd[$l - 1]);
    }

    echo "\n";
}

// Wrapper Function
function wrapper(array $arr, $n, array $l,
                            array $r, $q)
{
    $prefodd=array_fill(0,100,0);
    $prefeven=array_fill(0,100,0);

    // Evaluating prefixodd
    // and prefixeven
    for ($i = 1; $i <= $n; $i++)
    {
        if (($i) % 2 == 0)
        {
            $prefeven[$i] = $arr[$i - 1] ^
                        $prefeven[$i - 1];
            $prefodd[$i] = $prefodd[$i - 1];
        }
        else {
            $prefeven[$i] = $prefeven[$i - 1];
            $prefodd[$i] = $prefodd[$i - 1] ^
                                $arr[$i - 1];
        }
    }

    $i = 0;
    while ($i != $q) {
        ansQueries($prefeven, $prefodd,
                    $l[$i], $r[$i]);
        $i++;
    }
}

    // Driver code
    $arr = array ( 1, 2, 3, 4, 5 );
```

```
    $n = sizeof($arr) / sizeof($arr[0]);

    $l=array ( 1, 1, 2 );
    $r=array ( 2, 3, 4 );
    $q = sizeof($l) / sizeof($l[0]);

    wrapper($arr, $n, $l, $r, $q);

// This code is contributed by mits
?>
```

Output:

```
0
2
6
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/queries-on-xor-of-xors-of-all-subarrays/

Chapter 276

Quotient and remainder dividing by 2^k (a power of 2)

Quotient and remainder dividing by 2^k (a power of 2) - GeeksforGeeks

You are given an positive integer n as dividend and another number m (form of 2^k), you have to find quotient and remainder without performing actual division.

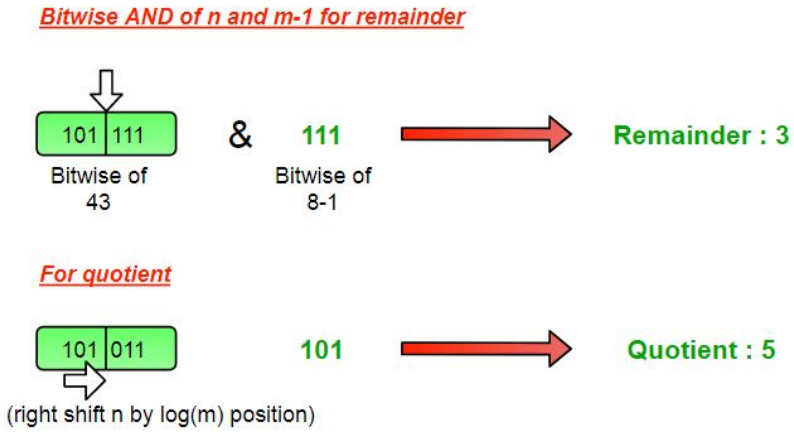
Examples:

```
Input : n = 43, m = 8
Output : Quotient = 5, Remainder = 3

Input : n = 58, m = 16
Output : Quotient = 3, Remainder = 10
```

In this we are using bitwise representation of a number for understanding the role of division of any number by divisor of form 2^k. All numbers which are power of two includes only 1 set bits in their representation and we will use this property.
For finding remainder we will take logical AND of the dividend (n) and divisor minus 1 (m-1), this will give only the set bits of dividend right to the set bit of divisor which is our actual remainder in that case.
Further, the left part of the dividend (from the position of set bit in divisor) would be considered for quotient. So, from dividend (n) removing all bits right from the position of set bit of divisor will result into quotient, and right shifting the dividend log2(m) times will do this job for finding the quotient.

- **Remainder = n & (m-1)**
- **Quotient = (n >> log2(m))**

Note : Log2(m) will give the number of bits present in the binary representation of m.



C++

```
 // CPP to find remainder and quotient
#include<bits/stdc++.h>
using namespace std;

// function to print remainder and quotient
void divide(int n,int m)
{
    // print Remainder by
    // n AND (m-1)
    cout <<"Remainder = " << ((n) &(m-1));

    // print quotient by
    // right shifting n by (log2(m)) times
    cout <<"\nQuotient = " <<(n >> (int)(log2(m)));

}

// driver program
int main()
{
    int n = 43, m = 8;
    divide(n, m);
    return 0;
}
```

Java

```
 // Java to find remainder and quotient
import java.io.*;
```

```
public class GFG {

    // function to print remainder and
    // quotient
    static void divide(int n, int m)
    {

        // print Remainder by
        // n AND (m-1)
        System.out.println("Remainder = "
                        + ((n) &(m-1)));

        // print quotient by right shifting
        // n by (log2(m)) times
        System.out.println("Quotient = "
            + (n >> (int)(Math.log(m) / Math.log(2))));
    }

    // driver program
    static public void main (String[] args)
    {
        int n = 43, m = 8;

        divide(n, m);
    }
}

// This code is contributed by vt_m.
```

Python 3

```
 # Python 3 to find remainder and
# quotient
import math
# function to print remainder and
# quotient
def divide(n, m):

    # print Remainder by
    # n AND (m-1)
    print("Remainder = ",
                  ((n) &(m-1)))

    # print quotient by
    # right shifting n by
    # (log2(m)) times
    print("Quotient = " ,(n >>
          (int)(math.log2(m))))
```

```
# driver program
n = 43
m = 8
divide(n, m)

# This code is contributed by
# Smitha
```

C#

```
 // C# to find remainder and quotient
using System;

public class GFG
{
    // function to print remainder and quotient
    static void divide(int n,int m)
    {
        // print Remainder by
        // n AND (m-1)
        Console.WriteLine("Remainder = " +((n) & (m - 1)));

        // print quotient by
        // right shifting n by (log2(m)) times
        Console.WriteLine("Quotient = "
                          + (n >> (int)(Math.Log(m))));

    }

    // Driver program
    static public void Main ()
    {
        int n = 43, m = 8;
        divide(n, m);

    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Code to find remainder
// and quotient
```

```
// function to print remainder
// and quotient
function divide($n,$m)
{

    // print Remainder by
    // n AND (m-1)
    echo "Remainder = ". (($n) &($m - 1));

    // print quotient by
    // right shifting n by
    // (log(m,2)) times 2
    // is base
    echo "\nQuotient = ".($n >> (int)(log($m, 2)));

}

    // Driver Code
    $n = 43;
    $m = 8;
    divide($n, $m);

//This code is contributed by mits
?>
```

Output:

```
Remainder = 3
Quotient = 5
```

Improved By : Mithun Kumar, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/quotient-remainder-dividing-2k-power-2/

Chapter 277

Range query for count of set bits

Range query for count of set bits - GeeksforGeeks

Given an array of positive integer and q query which contains two integers, L & R. Task is to find the number of set bits for a given range.

Prerequisite : Bitwise Hacks

Examples :

```
Input :  Arr[] = { 1, 5, 6, 10, 9, 4 }
         Query : 2
         L  &  R
         1     5
         2     4
Output : 9
         6

Input : Arr[] = { 1, 10, 5, 2, 8, 11, 15 }
        Query : 2
        L  &  R
        2     4
        1     5
Output :  4
          9
```

Simple solution to this problem is to run a loop from L to R and count number of set bits in a Range. This solution take O(nlog(s)) (where s is bits size) for each query.

Efficient solution is based on the fact that if we store count of all set bits of numbers in an array "BitCounts", then we answer each query in O(1) time. So, start traversing

the elements of array and count set bits for each element and store in array. Now, find cumulative sum of this array. This array will help in answering queries.

```
BitCount[] that will store the count of set bits
in a number.

Run a Loop from 0 to 31 "for 32 bits size integer "
-> mark elements with i'th bit set

Run an inner Loop from 0 to size of Array "Arr"
     -> Check whether the current bit is set or not
     -> if it's set then mark it.
          long  temp = arr[j] >> i;
             if (temp %2 != 0)
                 BitCount[j] += 1
```

Below is implementation of above idea.

C++

```
 // c++ program to Range query for
// Count number of set bits
#include <iostream>
using namespace std;

// 2-D array that will stored the count
// of bits set in element of array
int BitCount[10000] = { 0 };

// Function store the set bit
// count in BitCount Array
void fillSetBitsMatrix(int arr[], int n)
{

    // traverse over all bits
    for (int i = 0; i < 32; i++) {

        // mark elements with i'th bit set
        for (int j = 0; j < n; j++) {

            // Check whether the current bit is
            // set or not if it's set then mark it.
            long temp = arr[j] >> i;
            if (temp % 2 != 0)
                BitCount[j] += 1;
        }
```

```
    }

    // store cumulative sum of bits
    for (int i = 1; i < n; i++)
        BitCount[i] += BitCount[i - 1];
}

// Function to process queries
void Query(int Q[][2], int q)
{
    for (int i = 0; i < q; i++)
        cout << (BitCount[Q[i][1]] -
                 BitCount[Q[i][0] - 1]) << endl;
}

// Driver Code
int main()
{
    int Arr[] = { 1, 5, 6, 10, 9, 4, 67 };
    int n = sizeof(Arr) / sizeof(Arr[0]);

    fillSetBitsMatrix(Arr, n);

    int q = 2;
    int Q[2][2] = { { 1, 5 }, { 2, 6 } };

    Query(Q, q);

    return 0;
}
```

Java

```
 // java program to Range query for
// Count number of set bits
import java.io.*;

class GFG {

    // 2-D array that will stored the count
    // of bits set in element of array
    static int BitCount[] = new int[10000];

    // Function store the set bit
    // count in BitCount Array
    static void fillSetBitsMatrix(int arr[], int n)
    {
```

```
        // traverse over all bits
        for (int i = 0; i < 32; i++) {

            // mark elements with i'th bit set
            for (int j = 0; j < n; j++) {

                // Check whether the current
                // bit is set or not if it's
                // set then mark it.
                long temp = arr[j] >> i;
                if (temp % 2 != 0)
                    BitCount[j] += 1;
            }
        }

        // store cumulative sum of bits
        for (int i = 1; i < n; i++)
            BitCount[i] += BitCount[i - 1];
    }

    // Function to process queries
    static void Query(int Q[][], int q)
    {
        for (int i = 0; i < q; i++)
            System.out.println( (BitCount[Q[i][1]]
                        - BitCount[Q[i][0] - 1]));
    }

    // Driver Code
    public static void main (String[] args)
    {
        int Arr[] = { 1, 5, 6, 10, 9, 4, 67 };
        int n = Arr.length;

        fillSetBitsMatrix(Arr, n);

        int q = 2;
        int Q[][] = { { 1, 5 }, { 2, 6 } };

        Query(Q, q);
    }
}

// This code is contributed by anuj_67.
```

C#

```
 // C# program to Range query for
```

```
// Count number of set bits
using System;

class GFG {

    // 2-D array that will stored the count
    // of bits set in element of array
    static int []BitCount = new int[10000];

    // Function store the set bit
    // count in BitCount Array
    static void fillSetBitsMatrix(int []arr, int n)
    {

        // traverse over all bits
        for (int i = 0; i < 32; i++) {

            // mark elements with i'th bit set
            for (int j = 0; j < n; j++) {

                // Check whether the current
                // bit is set or not if it's
                // set then mark it.
                long temp = arr[j] >> i;
                if (temp % 2 != 0)
                    BitCount[j] += 1;
            }
        }

        // store cumulative sum of bits
        for (int i = 1; i < n; i++)
            BitCount[i] += BitCount[i - 1];
    }

    // Function to process queries
    static void Query(int [,]Q, int q)
    {
        for (int i = 0; i < q; i++)
            Console.WriteLine( (BitCount[Q[i,1]]
                        - BitCount[Q[i,0] - 1]));
    }

    // Driver Code
    public static void Main ()
    {
        int []Arr = { 1, 5, 6, 10, 9, 4, 67 };
        int n = Arr.Length;
```

```
        fillSetBitsMatrix(Arr, n);

        int q = 2;
        int [,]Q = { { 1, 5 }, { 2, 6 } };

        Query(Q, q);
    }
}

// This code is contributed by anuj_67.
```

Output:

```
9
10
```

Time Complexity : O(1) for each query.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/range-query-for-count-of-set-bits/

Chapter 278

Remove duplicates from a string in O(1) extra space

Remove duplicates from a string in O(1) extra space - GeeksforGeeks

Given a string *str* of lowercase characters, the task is to remove duplicates and return a resultant string without modifying the order of characters in the original string.

Examples:

```
Input: str = "geeksforgeeks"
Output: geksfor

Input: str = "characters"
Output: chartes
```

Approach: The idea is to use bits of a *counter* variable to mark the presence of a character in the string. To mark the presence of 'a' set 0th bit as 1, for 'b' set 1st bit as 1 and so on. If the corresponding bit of character present in original string is set to 0, it means it is the first occurrence of that character, hence set its corresponding bit as 1 and keep on including the current character in resultant string.

Consider the string str = "geeksforgeeks"

- character: 'g'
  x = 6(ascii of g – 97)
  6th bit in counter is unset resulting first occurrence of character 'g'.
  str[0] = 'g'
  counter = 00000000000000000000000001000000 // mark 6th bit as visited
  length = 1

- character: 'e'
  x = 4(ascii of e – 97)
  4th bit in counter is unset resulting in first occurrence of character 'e'.
  str[1] = 'e'
  counter = 00000000000000000000000001010000 //mark 4th bit as visited
  length = 2
- character: 'e'
  x = 4(ascii of e – 97)
  4th bit in counter is set resulting in duplicate character.
  Ignore this character. Move for next character.
  counter = 00000000000000000000000001010000 //same as previous
  length = 2
- character: 'k'
  x = 10(ascii of k – 97)
  10th bit in counter is unset resulting in first occurrence of character 'k'.
  str[2] = 'k'
  counter = 00000000000000000000010001010000 //mark 10th bit as visited
  length = 3

Similarly, do the same for all characters.

Resultant string : geksfor(string of length 7 starting from index 0)

Algorithm:

1. Initialize a counter variable (keeps track of the characters visited in string), it is a 32 bit Integer represented as(00000000000000000000000000000000) initially.
2. Consider 'a' as 0th bit of counter, 'b' as 1st bit of counter, 'c' as 2nd bit of counter and so on.
3. Traverse through each character of input string.
4. Get the character's value, where character's value(x) = Ascii of character – 97. This will make sure for value of 'a' as 0, value of 'b' as 1 and so on.
5. Check xth bit of counter.
6. If Xth bit of counter is 0 which means the current character has appeared for the first time, keep the current character at the index "length" of string .
7. Mark the current character visited by setting xth bit of counter.
8. Increment length.
9. Return Substring of size "length" from index 0.

Below is the implementation of above approach:

```
 // C++ implementation of above approach
#include <bits/stdc++.h>
#include <string>
using namespace std;

// Function to remove duplicates
string removeDuplicatesFromString(string str)
```

```
{

    // keeps track of visited characters
    int counter = 0;

    int i = 0;
    int size = str.size();

    // gets character value
    int x;

    // keeps track of length of resultant string
    int length = 0;

    while (i < size) {
        x = str[i] - 97;

        // check if Xth bit of counter is unset
        if ((counter & (1 << x)) == 0) {

            str[length] = 'a' + x;

            // mark current character as visited
            counter = counter | (1 << x);

            length++;
        }
        i++;
    }

    return str.substr(0, length);
}

// Driver program to test above function
int main()
{
    string str = "geeksforgeeks";
    cout << removeDuplicatesFromString(str);
    return 0;
}
```

Output:

```
geksfor
```

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

Source

https://www.geeksforgeeks.org/remove-duplicates-from-a-string-in-o1-extra-space/

Chapter 279

Reverse actual bits of the given number

Reverse actual bits of the given number - GeeksforGeeks

Given a non-negative integer **n**. The problem is to reverse the bits of **n** and print the number obtained after reversing the bits. Note that the actual binary representation of the number is being considered for reversing the bits, no leading 0's are being considered.

Examples :

```
Input : 11
Output : 13
(11)10 = (1011)2.
After reversing the bits we get:
(1101)2 = (13)10.

Input : 10
Output : 5
(10)10 = (1010)2.
After reversing the bits we get:
(0101)2 = (101)2
        = (5)10.
```

In this approach, one by one bits in binary representation of **n** are being obtained with the help of bitwise right shift operation and they are being accumulated in **rev** with the help of bitwise left shift operation.
Algorithm:

```
reverseBits(n)
    Initialize rev = 0

    while (n > 0)
        rev <<= 1
        if n & 1 == 1, then
            rev ^= 1
        n >>= 1

    return rev
```

C++

```
 // C++ implementation to reverse bits of a number
#include <bits/stdc++.h>

using namespace std;

// function to reverse bits of a number
unsigned int reverseBits(unsigned int n)
{
    unsigned int rev = 0;

    // traversing bits of 'n' from the right
    while (n > 0)
    {
        // bitwise left shift
        // 'rev' by 1
        rev <<= 1;

        // if current bit is '1'
        if (n & 1 == 1)
            rev ^= 1;

        // bitwise right shift
        // 'n' by 1
        n >>= 1;

    }

    // required number
    return rev;
}

// Driver program to test above
int main()
```

```
{
    unsigned int n = 11;
    cout << reverseBits(n);
    return 0;
}
```

Java

```
 // Java implementation to
// reverse bits of a number
class GFG
{
    // function to reverse bits of a number
    public static int reverseBits(int n)
    {
        int rev = 0;

        // traversing bits of 'n'
        // from the right
        while (n > 0)
        {
            // bitwise left shift
            // 'rev' by 1
            rev <<= 1;

            // if current bit is '1'
            if ((int)(n & 1) == 1)
                rev ^= 1;

            // bitwise right shift
            //'n' by 1
            n >>= 1;
        }
        // required number
        return rev;
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 11;
        System.out.println(reverseBits(n));
    }
}

// This code is contributed
// by  prerna saini.
```

Python 3

```
 # Python 3 implementation to
# reverse bits of a number


# function to reverse
# bits of a number
def reverseBits(n) :

    rev = 0

    # traversing bits of 'n' from the right
    while (n > 0) :

        # bitwise left shift 'rev' by 1
        rev = rev << 1

        # if current bit is '1'
        if (n & 1 == 1) :
            rev = rev ^ 1

        # bitwise right shift 'n' by 1
        n = n >> 1


    # required number
    return rev

# Driver code
n = 11
print(reverseBits(n))


# This code is contributed
# by Nikita Tiwari.
```

C#

```
 // C# implementation to
// reverse bits of a number
using System;
class GFG
{
    // function to reverse bits of a number
    public static int reverseBits(int n)
    {
```

```
        int rev = 0;

        // traversing bits of 'n'
        // from the right
        while (n > 0)
        {
            // bitwise left shift
            // 'rev' by 1
            rev <<= 1;

            // if current bit is '1'
            if ((int)(n & 1) == 1)
                rev ^= 1;

            // bitwise right shift
            //'n' by 1
            n >>= 1;
        }
        // required number
        return rev;
    }

    // Driver code
    public static void Main()
    {
        int n = 11;
        Console.WriteLine(reverseBits(n));
    }
}

// This code is contributed
// by vt_m.
```

PHP

```
 <?php
// PHP implementation to reverse
// bits of a number

// function to reverse
// bits of a number
function reverseBits($n)
{
    $rev = 0;

    // traversing bits of 'n'
    // from the right
    while ($n > 0)
```

```
    {
        // bitwise left shift
        // 'rev' by 1
        $rev <<= 1;

        // if current bit is '1'
        if ($n & 1 == 1)
            $rev ^= 1;

        // bitwise right shift
        // 'n' by 1
        $n >>= 1;

    }

    // required number
    return $rev;
}

// Driver code
$n = 11;
echo reverseBits($n);

// This code is contributed by mits
?>
```

Output :

```
13
```

Time Complexity: O(num), where **num** is the number of bits in the binary representation of **n**.

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/reverse-actual-bits-given-number/

Chapter 280

Reverse an array without using subtract sign '-' anywhere in the code

Reverse an array without using subtract sign '-' anywhere in the code - GeeksforGeeks

Given an array, the task is to reverse the array without using subtract sign '-' anywhere in your code. It is not tough to reverse an array but the main thing is to note use '-' operator.

Asked in: Moonfrog Interview

Below are different approaches:

Method 1:
1- Store array elements into a vector in C++.
2- Then reverse the vector using predefined functions.
3- Then store reversed elements into the array back.

Method 2:
1- Store array elements into a stack.
2- As the stack follows Last In First Out, so we can store elements from top of the stack into the array which will be itself in a reverse manner.

Method 3:
1- In this method the idea is to use negative sign but by storing it into a variable.
2- By using this statement x = (INT_MIN/INT_MAX), we get -1 in a variable x.
3- As INT_MIN and INT_MAX have same values just of opposite signs,so on dividing them it will give -1.
4- Then 'x' can be used in decrementing the index from last.

```
 // C++ program to reverse an array without
// using "-" sign
#include<bits/stdc++.h>
using namespace std;
```

```
// Function to reverse array
void reverseArray(int arr[], int n)
{
    // Trick to assign -1 to a variable
    int x = (INT_MIN/INT_MAX);

    // Reverse array in simple manner
    for (int i = 0; i<n/2 ; i++)

        // Swap ith index value with (n-i-1)th
        // index value
        swap(arr[i], arr[n + (x*i) + x]);
}

//Drivers code
int main()
{
    int arr[] = {5, 3, 7, 2, 1, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    reverseArray(arr, n);

    // print the reversed array
    for (int i=0; i< n ; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Output:

```
6 1 2 7 3 5
```

Method 4:
In this method 4, the idea is to use bitwise operator to implement subtraction i.e.
A – B = A + ~B + 1
so, i– can be written as i = i +~1 +1

```
 // C++ program to reverse an array without
// using "-" sign
#include<bits/stdc++.h>
using namespace std;

// Function to reverse array
void reverseArray(int arr[], int n)
```

```
{
    int x = (INT_MIN/INT_MAX);

    // Reverse array in simple manner
    for (int i = 0; i<n/2 ; i++)

        // Swap ith index value with (n-i-1)th
        // index value
        // Note : A - B = A + ~B + 1
        // So n - i = n + ~i + 1 then
        //    n - i - 1 = (n + ~i + 1) + ~1 + 1
        swap(arr[i], arr[(n + ~i + 1) + ~1 + 1]);
}

// Driver code
int main()
{
    int arr[] = {5, 3, 7, 2, 1, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    reverseArray(arr, n);

    // print the reversed array
    for (int i=0; i< n ; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Output:

```
6 1 2 7 3 5
```

Source

https://www.geeksforgeeks.org/reverse-array-without-using-subtract-sign-anywhere-code/

Chapter 281

Reverse bits using lookup table in O(1) time

Reverse bits using lookup table in O(1) time - GeeksforGeeks

Given an unsigned integer, reverse all bits of it and return the number with reversed bits.

Examples:

```
Input : n = 1
Output : 2147483648
On a machine with size of unsigned
bit as 32. Reverse of 0....001 is
100....0.

Input : n = 2147483648
Output : 1
```

In the previous post we had seen two method that solved this problem in O(n) & O(logn) time. Here we solve this problem in O(1) time using lookup table. It's hard to reverse all 32 bits (assuming this as size of int) in one go using lookup table (" because it's infeasible to create lookup table of size 2^{32}-1 "). So we break 32 bits into 8 bits of chunks(lookup table of size 2^8-1 "0-255").

Lookup Table
in lookup tale we will store reverse of every number that are in a range(0-255)

LookupTable[0] = 0 | binary 00000000 Reverse 00000000
LookupTable[1] = 128 | binary 00000001 reverse 10000000
LookupTable[2] = 64 | binary 00000010 reverse 01000000
LookupTanle[3] = 192 | binary 00000011 reverse 11000000
LookupTable[4] = 32 | binary 00000100 reverse 00100000
and so on… upto lookuptable[255].

Let's take an Example How lookup table work.
let number = 12456
in Binary = 00000000000000000011000010101000

```
Split it into 8 bits chunks  :  00000000 | 00000000 | 00110000 | 10101000
         in decimal          :     0          0          48       168
reverse each chunks using lookup table :
Lookuptable[ 0 ] = 0  | in binary 00000000
Lookuptable[48 ] = 12 | in binary 00001100
Lookuptable[168] = 21 | in binary 00010101

Now Binary :
00000000 | 00000000 | 00001100 | 00010101

Binary chunks after rearrangement :
00010101 | 00001100 | 00000000 | 00000000

Reverse of 12456 is 353107968
```

```
 // CPP program to reverse bits using lookup table.
#include<bits/stdc++.h>
using namespace std;

// Generate a lookup table for 32bit operating system
// using macro
#define R2(n)     n,     n + 2*64,     n + 1*64,     n + 3*64
#define R4(n) R2(n), R2(n + 2*16), R2(n + 1*16), R2(n + 3*16)
#define R6(n) R4(n), R4(n + 2*4 ), R4(n + 1*4 ), R4(n + 3*4 )

// Lookup table that store the reverse of each table
unsigned int lookuptable[256] = { R6(0), R6(2), R6(1), R6(3) };

/* Function to reverse bits of num */
int reverseBits(unsigned int num)
{
    int reverse_num = 0;

     // Reverse and then rearrange

                  // first chunk of 8 bits from right
     reverse_num = lookuptable[ num & 0xff ]<<24 |

                  // second chunk of 8 bits from  right
                  lookuptable[ (num >> 8) & 0xff ]<<16 |

                  lookuptable[ (num >> 16 )& 0xff ]<< 8 |
                  lookuptable[ (num >>24 ) & 0xff ] ;
```

```
    return reverse_num;
}

//driver program to test above function
int main()
{
    int x = 12456;
    printf("%u", reverseBits(x));
    return 0;
}
```

Output:

```
353107968
```

Time complexity :O(1)

Source

https://www.geeksforgeeks.org/reverse-bits-using-lookup-table-in-o1-time/

Chapter 282

Rotate bits of a number

Rotate bits of a number - GeeksforGeeks

Bit Rotation: A rotation (or circular shift) is an operation similar to shift except that the bits that fall off at one end are put back to the other end.

In left rotation, the bits that fall off at left end are put back at right end.

In right rotation, the bits that fall off at right end are put back at left end.

Example:
Let n is stored using 8 bits. Left rotation of n = 11100101 by 3 makes n = 00101111 (Left shifted by 3 and first 3 bits are put back in last). If n is stored using 16 bits or 32 bits then left rotation of n (000…11100101) becomes 00..00**11100101**000.
Right rotation of n = 11100101 by 3 makes n = 10111100 (Right shifted by 3 and last 3 bits are put back in first) if n is stored using 8 bits. If n is stored using 16 bits or 32 bits then right rotation of n (000…11100101) by 3 becomes **101**000..00**11100**.

C

```
 #include<stdio.h>
#define INT_BITS 32

/*Function to left rotate n by d bits*/
int leftRotate(int n, unsigned int d)
{
   /* In n<<d, last d bits are 0. To put first 3 bits of n at
     last, do bitwise or of n<<d with n >>(INT_BITS - d) */
   return (n << d)|(n >> (INT_BITS - d));
}

/*Function to right rotate n by d bits*/
int rightRotate(int n, unsigned int d)
{
   /* In n>>d, first d bits are 0. To put last 3 bits of at
```

```
     first, do bitwise or of n>>d with n <<(INT_BITS - d) */
   return (n >> d)|(n << (INT_BITS - d));
}

/* Driver program to test above functions */
int main()
{
  int n = 16;
  int d = 2;
  printf("Left Rotation of %d by %d is ", n, d);
  printf("%d", leftRotate(n, d));
  printf("\nRight Rotation of %d by %d is ", n, d);
  printf("%d", rightRotate(n, d));
  getchar();
}
```

Java

```
 // Java code to rotate bits
// of number
class GFG
{
static final int INT_BITS = 32;

/*Function to left rotate n by d bits*/
static int leftRotate(int n, int d) {

    /* In n<<d, last d bits are 0.
       To put first 3 bits of n at
       last, do bitwise or of n<<d with
       n >>(INT_BITS - d) */
    return (n << d) | (n >> (INT_BITS - d));
}

/*Function to right rotate n by d bits*/
static int rightRotate(int n, int d) {

    /* In n>>d, first d bits are 0.
       To put last 3 bits of at
       first, do bitwise or of n>>d
       with n <<(INT_BITS - d) */
    return (n >> d) | (n << (INT_BITS - d));
}

// Driver code
public static void main(String arg[])
{
    int n = 16;
```

```
    int d = 2;
    System.out.print("Left Rotation of " + n +
                         " by " + d + " is ");
    System.out.print(leftRotate(n, d));

    System.out.print("\nRight Rotation of " + n +
                            " by " + d + " is ");
    System.out.print(rightRotate(n, d));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 code to
# rotate bits of number

INT_BITS = 32

# Function to left
# rotate n by d bits
def leftRotate(n, d):

    # In n<<d, last d bits are 0.
    # To put first 3 bits of n at
    # last, do bitwise or of n<<d
    # with n >>(INT_BITS - d)
    return (n << d)|(n >> (INT_BITS - d))

# Function to right
# rotate n by d bits
def rightRotate(n, d):

    # In n>>d, first d bits are 0.
    # To put last 3 bits of at
    # first, do bitwise or of n>>d
    # with n <<(INT_BITS - d)
    return (n >> d)|(n << (INT_BITS - d)) & 0xFFFFFFFF

# Driver program to
# test above functions
n = 16
d = 2

print("Left Rotation of",n,"by"
      ,d,"is",end=" ")
print(leftRotate(n, d))
```

```
print("Right Rotation of",n,"by"
     ,d,"is",end=" ")
print(rightRotate(n, d))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to rotate
// bits of a number
using System;

class GFG
{
    static int INT_BITS = 32;

    /* Function to left rotate n by d bits*/
    static int leftRotate(int n, int d) {

        /* In n<<d, last d bits are 0.
        To put first 3 bits of n at
        last, do bitwise or of n<<d with
        n >>(INT_BITS - d) */
        return (n << d) | (n >> (INT_BITS - d));
    }

    /*Function to right rotate n by d bits*/
    static int rightRotate(int n, int d) {

        /* In n>>d, first d bits are 0.
        To put last 3 bits of at
        first, do bitwise or of n>>d
        with n <<(INT_BITS - d) */
        return (n >> d) | (n << (INT_BITS - d));
    }

    // Driver code
    public static void Main()
    {
        int n = 16;
        int d = 2;

        Console.Write("Left Rotation of " + n
                      + " by " + d + " is ");
        Console.Write(leftRotate(n, d));
```

```
        Console.Write("\nRight Rotation of " + n
                      + " by " + d + " is ");
        Console.Write(rightRotate(n, d));
    }
}

// This code is contributed by Sam007
```

Output :

```
Left Rotation of 16 by 2 is 64
Right Rotation of 16 by 2 is 4
```

Source

https://www.geeksforgeeks.org/rotate-bits-of-an-integer/

Chapter 283

Russian Peasant (Multiply two numbers using bitwise operators)

Russian Peasant (Multiply two numbers using bitwise operators) - GeeksforGeeks

Given two integers, write a function to multiply them without using multiplication operator.

There are many other ways to multiply two numbers (For example, see this). One interesting method is the Russian peasant algorithm. The idea is to double the first number and halve the second number repeatedly till the second number doesn't become 1. In the process, whenever the second number become odd, we add the first number to result (result is initialized as 0)
The following is simple algorithm.

```
Let the two given numbers be 'a' and 'b'
1) Initialize result 'res' as 0.
2) Do following while 'b' is greater than 0
   a) If 'b' is odd, add 'a' to 'res'
   b) Double 'a' and halve 'b'
3) Return 'res'.
```

C/C++

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

// A method to multiply two numbers using Russian Peasant method
unsigned int russianPeasant(unsigned int a, unsigned int b)
{
```

```
    int res = 0;  // initialize result

    // While second number doesn't become 1
    while (b > 0)
    {
         // If second number becomes odd, add the first number to result
         if (b & 1)
             res = res + a;

         // Double the first number and halve the second number
         a = a << 1;
         b = b >> 1;
     }
     return res;
}

// Driver program to test above function
int main()
{
    cout << russianPeasant(18, 1) << endl;
    cout << russianPeasant(20, 12) << endl;
    return 0;
}
```

Java

```
 // Java program for Russian Peasant Multiplication
import java.io.*;

class GFG
{
    // Function to multiply two
    // numbers using Russian Peasant method
    static int russianPeasant(int a, int b)
    {
        // initialize result
        int res = 0;

        // While second number doesn't become 1
        while (b > 0)
        {
             // If second number becomes odd,
             // add the first number to result
             if ((b & 1) != 0)
                 res = res + a;

            // Double the first number
            // and halve the second number
```

```
            a = a << 1;
            b = b >> 1;
        }
        return res;
    }

    // driver program
    public static void main (String[] args)
    {
        System.out.println(russianPeasant(18, 1));
        System.out.println(russianPeasant(20, 12));
    }
}

// Contributed by Pramod Kumar
```

Python 3

```
 # A method to multiply two numbers
# using Russian Peasant method

# Function to multiply two numbers
# using Russian Peasant method
def russianPeasant(a, b):

    res = 0 # initialize result

    # While second number doesn't
    # become 1
    while (b > 0):

        # If second number becomes
        # odd, add the first number
        # to result
        if (b & 1):
            res = res + a

        # Double the first number
        # and halve the second
        # number
        a = a << 1
        b = b >> 1

    return res

# Driver program to test
# above function
print(russianPeasant(18, 1))
```

```
print(russianPeasant(20, 12))
# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program for Russian Peasant Multiplication
using System;

class GFG {

    // Function to multiply two
    // numbers using Russian Peasant method
    static int russianPeasant(int a, int b)
    {
        // initialize result
        int res = 0;

        // While second number doesn't become 1
        while (b > 0) {

            // If second number becomes odd,
            // add the first number to result
            if ((b & 1) != 0)
                res = res + a;

            // Double the first number
            // and halve the second number
            a = a << 1;
            b = b >> 1;
        }
        return res;
    }

    // driver program
    public static void Main()
    {
        Console.WriteLine(russianPeasant(18, 1));
        Console.WriteLine(russianPeasant(20, 12));
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
```

```
// PHP Code to multiply two numbers
// using Russian Peasant method

// function returns the result
function russianPeasant($a, $b)
{

    // initialize result
    $res = 0;

    // While second number
    // doesn't become 1
    while ($b > 0)
    {

        // If second number becomes odd,
        // add the first number to result
        if ($b & 1)
            $res = $res + $a;

        // Double the first number and
        // halve the second number
        $a = $a << 1;
        $b = $b >> 1;
    }
    return $res;
}

    // Driver Code
    echo russianPeasant(18, 1), "\n";
    echo russianPeasant(20, 12), "\n";

// This code is contributed by Ajit
?>
```

Output:

```
18
240
```

How does this work?
The value of a*b is same as (a*2)*(b/2) if b is even, otherwise the value is same as ((a*2)*(b/2) + a). In the while loop, we keep multiplying 'a' with 2 and keep dividing 'b' by 2. If 'b' becomes odd in loop, we add 'a' to 'res'. When value of 'b' becomes 1, the value of 'res' + 'a', gives us the result.
Note that when 'b' is a power of 2, the 'res' would remain 0 and 'a' would have the multiplication. See the reference for more information.

Reference:
http://mathforum.org/dr.math/faq/faq.peasant.html

This article is compiled by **Shalki Agarwal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : Smitha Dinesh Semwal, jit_t

Source

https://www.geeksforgeeks.org/russian-peasant-multiply-two-numbers-using-bitwise-operators/

Chapter 284

Same Number Of Set Bits As N

Same Number Of Set Bits As N - GeeksforGeeks

Given a positive integer N, find out how many positive integers strictly less than N have the same number of set bits as N.

Examples :

```
Input : 8
Output :3
Explanation: Binary representation of
8 : 1000, so number of set bits in 8 is 1.
So the integers less than 8 with same number
of set bits are : 4, 2, 1

Input :1
Output :0

Input :4
Output :2
```

Approach:

```
1. Using __builtin_popcount() inbuilt function, count set bits in N and store into a
temp variable
2. Iterate from n-1 to 1 and also count set bits in i using __builtin_popcount()
function
3. Now, compare temp with __builtin_popcount(i)
4. If both are equal then increment counter variable
5. Return counter
```

Below is the implementation of above approach.

C++

```
 // CPP program to find numbers less than N
// that have same Number Of Set Bits As N
#include <iostream>
using namespace std;

int smallerNumsWithSameSetBits(int n)
{
    // __builtin_popcount function that count
    // set bits in n
    int temp = __builtin_popcount(n);

    // Iterate from n-1 to 1
    int count = 0;
    for (int i = n - 1; i > 0; i--) {

        // check if the number of set bits
        // equals to temp increment count
        if (temp == __builtin_popcount(i))
            count++;
    }
    return count;
}

// Driver Code
int main()
{
    int n = 4;
    cout << smallerNumsWithSameSetBits(n);
    return 0;
}
```

Java

```
 // Java program to find numbers less than N
// that have same Number Of Set Bits As N
class GFG {

    // returns number of set bits in a number
    static int __builtin_popcount(int n)
    {
        int d, t = 0;

        while(n > 0)
```

```
        {
            d = n % 2;
            n = n / 2;
            if(d == 1)
                t++;
        }
        return t;
    }

    static int smallerNumsWithSameSetBits(int n)
    {
        // __builtin_popcount function that count
        // set bits in n
        int temp = __builtin_popcount(n);

        // Iterate from n-1 to 1
        int count = 0;
        for (int i = n - 1; i > 0; i--) {

            // check if the number of set bits
            // equals to temp increment count
            if (temp == __builtin_popcount(i))
                count++;
        }
        return count;
    }

    // Driver Code
    public static void main(String[] args)
    {
        int n = 4;
        System.out.println(
            smallerNumsWithSameSetBits(n));
    }
}

// This code is contributed by Arnab Kundu.
```

Python3

```
 # Python3 program to find numbers
# less than N that have same
# Number Of Set Bits As N
def __builtin_popcount(n) :
    t = 0
    while(n > 0) :
        d = n % 2
        n = int(n / 2)
```

```
        if(d == 1) :
            t = t + 1
    return t

def smallerNumsWithSameSetBits(n) :

    # __builtin_popcount function
    # that count set bits in n
    temp = __builtin_popcount(n)

    # Iterate from n-1 to 1
    count = 0
    for i in range(n-1,0,-1) :

        # check if the number of
        # set bits equals to temp
        # increment count
        if (temp == __builtin_popcount(i)) :
            count = count + 1
    return count

# Driver Code
n = 4
print (smallerNumsWithSameSetBits(n))

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# program to find numbers less than N
// that have same Number Of Set Bits As N
using System;

class GFG {

    // returns number of set bits in a number
    static int __builtin_popcount(int n)
    {
        int d, t = 0;

        while(n > 0)
        {
            d = n % 2;
            n = n / 2;
            if(d == 1)
                t++;
        }
```

```
        return t;
    }

    static int smallerNumsWithSameSetBits(int n)
    {
        // __builtin_popcount function that count
        // set bits in n
        int temp = __builtin_popcount(n);

        // Iterate from n-1 to 1
        int count = 0;
        for (int i = n - 1; i > 0; i--) {

            // check if the number of set bits
            // equals to temp increment count
            if (temp == __builtin_popcount(i))
                count++;
        }
        return count;
    }

    // Driver Code
    static public void Main(String []args)
    {
        int n = 4;
        Console.WriteLine(
            smallerNumsWithSameSetBits(n));
    }
}

// This code is contributed by Arnab Kundu.
```

PHP

```
 <?php
// PHP program to find numbers
// less than N that have same
// Number Of Set Bits As N
function __builtin_popcount($n)
{
    $t = 0;
    while($n > 0)
    {
        $d = $n % 2;
        $n = intval($n / 2);
        if($d == 1)
            $t++;
    }
```

```
    return $t;
}
function smallerNumsWithSameSetBits($n)
{
    // __builtin_popcount function
    // that count set bits in n
    $temp = __builtin_popcount($n);

    // Iterate from n-1 to 1
    $count = 0;
    for ($i = $n - 1; $i > 0; $i--)
    {

        // check if the number of
        // set bits equals to temp
        // increment count
        if ($temp == __builtin_popcount($i))
            $count++;
    }
    return $count;
}

// Driver Code
$n = 4;
echo (smallerNumsWithSameSetBits($n));

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output:

```
2
```

Improved By : andrew1234, manishshaw1

Source

https://www.geeksforgeeks.org/number-set-bits-n/

Chapter 285

Set all even bits of a number

Set all even bits of a number - GeeksforGeeks

Given a number, the task is to set all even bit of a number. Positions of bits are counted from LSB (least significant bit) to MSB (Most significant bit). Position of LSB is considered as 1.

Examples :

```
Input : 20
Output : 30
Binary representation of 20 is
10100. After setting
even bits, we get 11110

Input : 10
Output : 10
```

Method 1:-
1. First generate a number that contains even position bits.
2. Take OR with the original number. Note that 1 | 1 = 1 and 1 | 0 = 1.

Let's understand this approach with below code.

C++

```
 // Simple CPP program to set all even
// bits of a number
#include <iostream>
using namespace std;

// Sets even bits of n and returns
// modified number.
```

```
int evenbitsetnumber(int n)
{
    // Generate 101010...10 number and
    // store in res.
    int count = 0, res = 0;
    for (int temp = n; temp > 0; temp >>= 1) {

        // if bit is even then generate
        // number and or with res
        if (count % 2 == 1)
            res |= (1 << count);

        count++;
    }

    // return OR number
    return (n | res);
}

int main()
{
    int n = 10;
    cout << evenbitsetnumber(n);
    return 0;
}
```

Java

```
 // Simple Java program to set all even
// bits of a number
class GFG
{
    // Sets even bits of n and returns
    // modified number.
    static int evenbitsetnumber(int n)
    {
        // Generate 101010...10 number and
        // store in res.
        int count = 0, res = 0;
        for (int temp = n; temp > 0; temp >>= 1) {

            // if bit is even then generate
            // number and or with res
            if (count % 2 == 1)
            res |= (1 << count);

        count++;
        }
```

```
        // return OR number
        return (n | res);

    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 4;
         System.out.println(evenbitsetnumber(n));
    }
}

// This code is contributed
// by  prerna saini.
```

Python3

```
 # Simple Python program to set all even
# bits of a number

# Sets even bits of n and returns
# modified number.
def evenbitsetnumber(n):

    # Generate 101010...10 number and
    # store in res.
    count = 0
    res = 0
    temp = n
    while(temp > 0):

        # if bit is even then generate
        # number and or with res
        if (count % 2 == 1):
            res |= (1 << count)

        count+=1
        temp >>= 1

    # return OR number
    return (n | res)

n = 10
print(evenbitsetnumber(n))

# This code is contributed
```

```
# by Smitha Dinesh Semwal
```

C#

```
 // Simple C# program to set
// all even bits of a number
using System;

class GFG
{
    // Sets even bits of n and
    // returns modified number.
    static int evenbitsetnumber(int n) {

        // Generate 101010...10 number
        //  and store in res.
        int count = 0, res = 0;
        for (int temp = n; temp > 0; temp >>= 1) {

            // if bit is even then generate
            // number and or with res
            if (count % 2 == 1)
            res |= (1 << count);

        count++;
        }

        // return OR number
        return (n | res);

    }

    // Driver code
    public static void Main()
    {
        int n = 4;
        Console.WriteLine(evenbitsetnumber(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// Simple php program to set
// all even bits of a number
```

```
// Sets even bits of n and
// returns  modified number.
function evenbitsetnumber($n)
{

    // Generate 101010...10 number
    // and store in res.
    $count = 0;
    $res = 0;
    for ($temp = $n; $temp > 0; $temp >>= 1)
    {

        // if bit is even then generate
        // number and or with res
        if ($count % 2 == 1)
            $res |= (1 << $count);

        $count++;
    }

    // return OR number
    return ($n | $res);
}

    //Driver Code
    $n = 10;
    echo evenbitsetnumber($n);

//This Code is contributed by mits
?>
```

Output:

```
10
```

Method 2 (A O(1) solution for 32 bit numbers)

C++

```
 // Efficient CPP program to set all even
// bits of a number
#include <iostream>
using namespace std;

// return msb set number
```

```
int getmsb(int n)
{

    // set all bits
    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // return msb
    // increment n by 1
    // and shift by 1
    return (n + 1) >> 1;
}

// return even seted number
int getevenbits(int n)
{

    // get msb here
    n = getmsb(n);

    // generate even bits like 101010..
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // if bits is odd then shift by 1
    if (n & 1)
        n = n >> 1;

    // return even set bits number
    return n;
}

// set all even bits here
int setallevenbits(int n)
{
    // take or with even set bits number
    return n | getevenbits(n);
}

int main()
{
    int n = 10;
    cout << setallevenbits(n);
```

```
    return 0;
}
```

Java

```
 // Efficient Java program to
// set all even bits of a number
import java.io.*;

class GFG
{

// return msb set number
static int getmsb(int n)
{

    // set all bits
    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // return msb
    // increment n by 1
    // and shift by 1
    return (n + 1) >> 1;
}

// return even seted number
static int getevenbits(int n)
{

    // get msb here
    n = getmsb(n);

    // generate even
    // bits like 101010..
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // if bits is odd
    // then shift by 1
    if ((n & 1) == 1)
        n = n >> 1;
```

```
    // return even
    // set bits number
    return n;
}

// set all even bits here
static int setallevenbits(int n)
{
    // take or with even
    // set bits number
    return n | getevenbits(n);
}

// Driver Code
public static void main (String[] args)
{
    int n = 10;
    System.out.println(setallevenbits(n));
}
}

// This code is contributed by ajit
```

Python3

```
 # Efficient Python 3
# program to set all even
# bits of a number

# return msb set number
def getmsb(n):

    # set all bits
    n |= n >> 1
    n |= n >> 2
    n |= n >> 4
    n |= n >> 8
    n |= n >> 16

    # return msb
    # increment n by 1
    # and shift by 1
    return (n + 1) >> 1


# return even seted number
def getevenbits(n):
```

```
    # get msb here
    n = getmsb(n)

    # generate even bits like 101010..
    n |= n >> 2
    n |= n >> 4
    n |= n >> 8
    n |= n >> 16

    # if bits is odd then shift by 1
    if (n & 1):
        n = n >> 1

    # return even set bits number
    return n


# set all even bits here
def setallevenbits(n):

    # take or with even set bits number
    return n | getevenbits(n)

n = 10
print(setallevenbits(n))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // Efficient C# program to
// set all even bits of a number
using System;

class GFG
{

// return msb set number
static int getmsb(int n)
{

    // set all bits
    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;
```

```
    // return msb
    // increment n by 1
    // and shift by 1
    return (n + 1) >> 1;
}

// return even seted number
static int getevenbits(int n)
{

    // get msb here
    n = getmsb(n);

    // generate even
    // bits like 101010..
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // if bits is odd
    // then shift by 1
    if ((n & 1) == 1)
        n = n >> 1;

    // return even
    // set bits number
    return n;
}

// set all even bits here
static int setallevenbits(int n)
{
    // take or with even
    // set bits number
    return n | getevenbits(n);
}

// Driver Code
public static void Main ()
{
    int n = 10;

    Console.WriteLine(setallevenbits(n));
}
}
```

```
// This code is contributed by aj_36
```

PHP

```
 <?php
// Efficient php program to set
// all even bits of a number

// return msb set number
function getmsb($n)
{

    // set all bits
    $n |= $n >> 1;
    $n |= $n >> 2;
    $n |= $n >> 4;
    $n |= $n >> 8;
    $n |= $n >> 16;

    // return msb
    // increment n by 1
    // and shift by 1
    return ($n + 1) >> 1;
}

// return even seted number
function getevenbits($n)
{

    // get msb here
    $n = getmsb($n);

    // generate even bits
    // like 101010..
    $n |= $n >> 2;
    $n |= $n >> 4;
    $n |= $n >> 8;
    $n |= $n >> 16;

    // if bits is odd then
    // shift by 1
    if ($n & 1)
        $n = $n >> 1;

    // return even set
    // bits number
    return $n;
}
```

```
// set all even bits here
function setallevenbits($n)
{
    // take or with even
    // set bits number
    return $n | getevenbits($n);
}

    //Driver code
    $n = 10;
    echo setallevenbits($n);

//This code is contributed by mits
?>
```

Output:

```
10
```

Improved By : Mithun Kumar, jit_t

Source

https://www.geeksforgeeks.org/set-all-even-bits-of-a-number/

Chapter 286

Set all odd bits of a number

Set all odd bits of a number - GeeksforGeeks

Given a number, the task is to set all odd bits of a number. Positions of bits are counted from LSB (least significant bit) to MSB (Most significant bit). Position of LSB is considered as 1.

Examples :

```
Input : 20
Output : 21
Explanation : Binary representation of 20
is 10100. Setting all odd
bits make the number 10101 which is binary
representation of 21.

Input : 10
Output : 15
```

Method 1 (Using XOR)
1. First generate a number that contains odd position bits.
2. Take OR with the original number. Note that $1 \mid 1 = 1$ and $1 \mid 0 = 1$.

Let's understand this approach with below code.

C++

```
 // CPP code Set all odd bits
// of a number
#include <iostream>
using namespace std;
```

```
// set all odd bit
int oddbitsetnumber(int n)
{
    int count = 0;

    // res for store 010101.. number
    int res = 0;

    // generate number form of 010101.....till
    // temp size
    for (int temp = n; temp > 0; temp >>= 1) {

        // if bit is odd, then generate
        // number and or with res
        if (count % 2 == 0)
            res |= (1 << count);

        count++;
    }

    return (n | res);
}

// Driver code
int main()
{
    int n = 10;
    cout << oddbitsetnumber(n);
    return 0;
}
```

Java

```
 // Java code to Set all odd
// bits of a number

class GFG
{

    // set all odd bit
    static int oddbitsetnumber(int n)
    {
        int count = 0;

        // res for store 010101.. number
        int res = 0;
```

```
        // generate number form of
        // 010101.....till temp size
        for (int temp = n; temp > 0; temp >>= 1)
        {

            // if bit is odd, then generate
            // number and or with res
            if (count % 2 == 0)
                res |= (1 << count);

            count++;
        }

        return (n | res);
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 10;
        System.out.println(oddbitsetnumber(n));
    }
}

// This code is contributed
// by prerna saini
```

Python3

```
 ''' Python3 code Set all odd bits
   of a number'''

# set all odd bit
def oddbitsetnumber(n):
    count = 0
    # res for store 010101.. number
    res = 0

    # generate number form of 010101.....till
    # temp size
    temp = n
    while temp > 0:

        # if bit is odd, then generate
        # number and or with res
        if count % 2 == 0:
            res |= (1 << count)
```

```
        count += 1
        temp >>= 1

    return (n | res)

n = 10
print (oddbitsetnumber(n))

#This code is contributed by Shreyanshi Arun.
```

C#

```
 // C# code to Set all odd
// bits of a number
using System;

class GFG
{
    static int oddbitsetnumber(int n)
    {
        int count = 0;

        // res for store 010101.. number
        int res = 0;

        // generate number form of
        // 010101.....till temp size
        for (int temp = n; temp > 0;
                           temp >>= 1)
        {

            // if bit is odd, then
            // generate number and
            // or with res
            if (count % 2 == 0)
                res |= (1 << count);

            count++;
        }

        return (n | res);
    }

    // Driver Code
    static public void Main ()
    {
        int n = 10;
        Console.WriteLine(oddbitsetnumber(n));
```

```
    }
}

// This code is contributed
// by prerna ajit
```

PHP

```
 <?php
// php code Set all odd
// bits  of a number

// set all odd bit
function oddbitsetnumber($n)
{
    $count = 0;

    // res for store 010101..
    // number
    $res = 0;

    // generate number form of
    // 010101... till temp size
    for ($temp = $n; $temp > 0; $temp >>= 1)
    {

        // if bit is odd, then generate
        // number and or with res
        if ($count % 2 == 0)
            $res |= (1 << $count);

        $count++;
    }

    return ($n | $res);
}

    // Driver code
    $n = 10;
    echo oddbitsetnumber($n);

// This code is contributed by mits
?>
```

Output:

```
15
```

Method 2 (A O(1) solution for 32 bit numbers)

C++

```
 // Efficient CPP program to set all
// odd bits number
#include <iostream>
using namespace std;

// return MSB set number
int getmsb(int n)
{
    // set all bits including MSB.
    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // return MSB
    return (n + 1) >> 1;
}

// Returns a number of same size (MSB at
// same position) as n and all odd bits
// set.
int getevenbits(int n)
{
    n = getmsb(n);

    // generate odd bits like 010101..
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // if bits is even then shift by 1
    if ((n&1) == 0)
        n = n >> 1;

    // return odd set bits number
    return n;
}

// set all odd bits here
int setalloddbits(int n)
```

```
{
    // take OR with odd set bits number
    return n | getevenbits(n);
}

// Driver code
int main()
{
    int n = 10;
    cout << setalloddbits(n);
    return 0;
}
```

Java

```
 // Efficient Java program to set
// all odd bits number
class GFG {

    // return MSB set number
    static int getmsb(int n)
    {
        // set all bits including MSB.
        n |= n >> 1;
        n |= n >> 2;
        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;

        // return MSB
        return (n + 1) >> 1;
    }

    // Returns a number of same
    // size (MSB at same position)
    // as n and all odd bits set.
    static int getevenbits(int n)
    {
        n = getmsb(n);

        // generate odd bits
        // like 010101..
        n |= n >> 2;
        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;

        // if bits is even
```

```
        // then shift by 1
        if ((n & 1) == 0)
            n = n >> 1;

        // return odd set bits number
        return n;
    }

    // set all odd bits here
    static int setalloddbits(int n)
    {
        // take OR with odd
        // set bits number
        return n | getevenbits(n);
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 10;
        System.out.println(setalloddbits(n));
    }
}

// This code is contributed
// by prerna saini
```

Python3

```
 # Efficient python3 program to
# set all odd bits number
import math

# return MSB set number
def getmsb( n):

    # set all bits including MSB.
    n |= n >> 1
    n |= n >> 2
    n |= n >> 4
    n |= n >> 8
    n |= n >> 16

    # return MSB
    return (n + 1) >> 1


# Returns a number of same
```

```
# size (MSB at same position)
# as n and all odd bits set.
def getevenbits(n):

    n = getmsb(n)

    # generate odd bits
    # like 010101..
    n |= n >> 2
    n |= n >> 4
    n |= n >> 8
    n |= n >> 16

    # if bits is even
    # then shift by 1
    if ((n & 1) == 0):
        n = n >> 1

    # return odd set bits number
    return n


# set all odd bits here
def setalloddbits( n):

    # take OR with odd
    # set bits number
    return n | getevenbits(n)


# Driver Program
n = 10
print(setalloddbits(n))


# This code is contributed
# by Gitanjali.
```

C#

```
 // Efficient C# program to
// set all odd bits number
using System;

class GFG
{

    // return MSB set number
```

```
static int getmsb(int n)
{
    // set all bits
    // including MSB.
    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // return MSB
    return (n + 1) >> 1;
}

// Returns a number of same
// size (MSB at same position)
// as n and all odd bits set.
static int getevenbits(int n)
{
    n = getmsb(n);

    // generate odd bits
    // like 010101..
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // if bits is even
    // then shift by 1
    if ((n & 1) == 0)
        n = n >> 1;

    // return odd
    // set bits number
    return n;
}

// set all odd bits here
static int setalloddbits(int n)
{
    // take OR with odd
    // set bits number
    return n | getevenbits(n);
}

// Driver Code
static public void Main ()
```

```
    {
        int n = 10;
        Console.WriteLine(setalloddbits(n));
    }
}

// This code is contributed ajit
```

PHP

```
 <?php
// Efficient php program to
// set all  odd bits number

// return MSB set number
function getmsb($n)
{

    // set all bits
    // including MSB.
    $n |= $n >> 1;
    $n |= $n >> 2;
    $n |= $n >> 4;
    $n |= $n >> 8;
    $n |= $n >> 16;

    // return MSB
    return ($n + 1) >> 1;
}

// Returns a number of
// same size (MSB at
// same position) as n
// and all odd bits set
function getevenbits($n)
{
    $n = getmsb($n);

    // generate odd bits
    // like 010101..
    $n |= $n >> 2;
    $n |= $n >> 4;
    $n |= $n >> 8;
    $n |= $n >> 16;

    // if bits is even
    // then shift by 1
    if (($n&1) == 0)
```

```
        $n = $n >> 1;

    // return odd set
    // bits number
    return $n;
}

// set all odd bits here
function setalloddbits($n)
{

    // take OR with odd
    // set bits number
    return $n | getevenbits($n);
}

    // Driver code
    $n = 10;
    echo setalloddbits($n);

// This code is contributed by mits
?>
```

Output:

```
15
```

Improved By : Mithun Kumar, jit_t

Source

https://www.geeksforgeeks.org/set-odd-bits-number/

Chapter 287

Set all the bits in given range of a number

Set all the bits in given range of a number - GeeksforGeeks

Given a non-negative number n and two values l and r. The problem is to set the bits in the range l to r in the binary representation of n, i.e, to unset bits from the rightmost lth bit to the rightmost r-th bit.
Constraint: 1 <= l <= r <= number of bits in the binary representation of n.

Examples :

```
Input : n = 17, l = 2, r = 3
Output : 23
(17)10 = (10001)2
(23)10 = (10111)2
The bits in the range 2 to 3 in the binary
representation of 17 are set.

Input : n = 50, l = 2, r = 5
Output : 62
```

Approach: Following are the steps:

```
1. Find a number 'range' that has all set
   bits in given range. And all other bits
   of this number are 0.
     range = (((1 << (l - 1)) - 1) ^
              ((1 << (r)) - 1));

2. Now, perform "n = n | range". This will
```

```
   set the bits in the range from l to r
   in n.
```

C++

```
 // C++ implementation to Set bits in
// the given range
#include <iostream>
using namespace std;

// function to toggle bits in the given range
int setallbitgivenrange(int n, int l, int r)
{
    // calculating a number 'range' having set
    // bits in the range from l to r and all other
    // bits as 0 (or unset).
    int range = (((1 << (l - 1)) - 1) ^
                ((1 << (r)) - 1));

    return (n | range);
}

// Driver code
int main()
{
    int n = 17, l = 2, r = 3;
    cout << setallbitgivenrange(n, l, r);
    return 0;
}
```

Java

```
 // java implementation to Set bits in
// the given range
import java.util.*;

class GFG
{

    // function to toggle bits in the
    // given range
    static int setallbitgivenrange(int n,
                               int l, int r)
    {

        // calculating a number 'range'
        // having set bits in the range
```

```
        // from l to r and all other
        // bits as 0 (or unset).
        int range = (((1 << (l - 1)) - 1) ^
                    ((1 << (r)) - 1));

        return (n | range);
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 17, l = 2, r = 3;

        System.out.println(setallbitgivenrange(
                                    n, l, r));
    }

}

// This code is contributed by Sam007.
```

Python3

```
 # Python3 implementation to Set
# bits in the given range

# Function to toggle bits
# in the given range
def setallbitgivenrange(n, l, r):

    # calculating a number 'range'
    # having set bits in the range
    # from l to r and all other
    # bits as 0 (or unset).
    range = (((1 << (l - 1)) - 1) ^
                ((1 << (r)) - 1))

    return (n | range)

# Driver code
n, l, r = 17, 2, 3
print(setallbitgivenrange(n, l, r))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# implementation to Set
```

```
// bits in the given range
using System;

class GFG
{
    // function to toggle bits
    // in the given range
    static int setallbitgivenrange(int n, int l, int r)
    {

        // calculating a number 'range'
        // having set bits in the range
        // from l to r and all other
        // bits as 0 (or unset).
        int range = (((1 << (l - 1)) - 1) ^
                     ((1 << (r)) - 1));

        return (n | range);
    }

    // Driver code
    static void Main()
    {
        int n = 17, l = 2, r = 3;
        Console.Write(setallbitgivenrange(n, l, r));
    }

}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP implementation to Set
// bits in the given range

// function to toggle bits
// in the given range
function setallbitgivenrange($n, $l, $r)
{
    // calculating a number 'range'
    // having set bits in the range
    // from l to r and all other
    // bits as 0 (or unset).
    $range = (((1 << ($l - 1)) - 1) ^
                  ((1 << ($r)) - 1));
```

```
    return ($n | $range);
}

// Driver code
$n = 17;
$l = 2;
$r = 3;
echo setallbitgivenrange($n, $l, $r);

// This code is contributed by Sam007
?>
```

Output :

```
23
```

Improved By : Sam007

Source

https://www.geeksforgeeks.org/set-bits-given-range-number/

Chapter 288

Set bits in N equals to M in the given range.

Set bits in N equals to M in the given range. - GeeksforGeeks

You are given two 32-bit numbers, N and M, and two bit positions, i and j. Write a method to set all bits between i and j in N equal to M (e.g., M becomes a substring of N located at i and starting at j).

Examples :

```
Input : N = 1, M = 2, i = 2, j = 4
Output: 9
N = 00000001(Considering 8 bits only)
M = 10 (Binary of 2) For more indexes,
leading zeroes will be considered.
Now set 3 bits from ith index to j in
the N as in the M.
Bits:-    0 0 0 (0  1  0) 0 1 = 9
Indexes:- 7 6 5  4  3  2  1 0
From index 2 to 4, bits are set according
to the M.
```

Asked in : Adobe

A simple solution is to traverse all bits in N from 0 to 31 and set the bits equals to M in the range from i to j.

An efficient solution is to do following steps.

1. Set all the bits after j in a number.

2. Set all the bits before i in a number.
3. Then perform Bitwise Or on both then we get the number with all the bits set except from i to j.
4. Perform Bitwise And with the given N as to set the bits according to the N.
5. Then shift M into the correct position i.e. in the range of i to j.
6. And at the last perform Bitwise Or on (Shifted M and the N modifed in 4th step).
7. The result will be N with M as substring from ith to jth bits

C++

```
 // C++ program for above implementation
#include <iostream>
using namespace std;

// Function to set the bits
int setBits(int n, int m, int i, int j)
{
    // number with all 1's
    int allOnes = ~0;

    // Set all the bits in the left of j
    int left = allOnes << (j + 1);

    // Set all the bits in the right of j
    int right = ((1 << i) - 1);

    // Do Bitwsie OR to get all the bits
    // set except in the range from i to j
    int mask = left | right;

    // clear bits j through i
    int masked_n = n & mask;

    // move m into the correct position
    int m_shifted = m << i;

    // return the Bitwise OR of masked_n
    // and shifted_m
    return (masked_n | m_shifted);
}

// Drivers program
int main()
{
    int n = 2, m = 4;
    int i = 2, j = 4;
    cout << setBits(n, m, i, j);
    return 0;
```

```
}
```

Java

```
 // Java Program
public class GFG
{
    // Function to set the bits
    static int setBits(int n, int m, int  i, int j)
    {

        // number with all 1's
        int  allOnes = ~0;

        // Set all the bits in the left of j
        int left = allOnes << (j + 1);

        // Set all the bits in the right of j
        int right = ((1 << i) - 1);

        // Do Bitwise OR to get all the bits
        // set except in the range from i to j
        int mask = left | right;

        // clear bits j through i
        int masked_n = n & mask;

        // move m into the correct position
        int m_shifted = m << i;

        // return the Bitwise OR of masked_n
        // and shifted_m
        return (masked_n | m_shifted);
    }

    // Driver Program to test above function
    public static void main(String[] args)
    {
        int n = 2, m = 4;
        int i = 2, j = 4;
        System.out.println(setBits(n, m, i, j));
    }
}
// This code is contributed by Sumit Ghosh
```

Python

```
 # Python program for above implementation
```

```
# Function to set the bits
def setBits(n, m, i, j):

    # number with all 1's
    allOnes = not 0

    # Set all the bits in the left of j
    left = allOnes << (j + 1)

    # Set all the bits in the right of j
    right = ((1 << i) - 1)

    # Do Bitwsie OR to get all the bits
    # set except in the range from i to j
    mask = left | right

    # clear bits j through i
    masked_n = n & mask

    # move m into the correct position
    m_shifted = m << i

    # return the Bitwise OR of masked_n
    # and shifted_m
    return (masked_n | m_shifted)

# Drivers program
n, m = 2, 4
i, j = 2, 4
print setBits(n, m, i, j)

# This code is submitted by Sachin Bisht
```

C#

```
 // C# Program for above implementation
using System;

public class GFG {

    // Function to set the bits
    static int setBits(int n, int m, int  i, int j)
    {

        // number with all 1's
        int  allOnes = ~0;
```

```
        // Set all the bits in the left of j
        int left = allOnes << (j + 1);

        // Set all the bits in the right of j
        int right = ((1 << i) - 1);

        // Do Bitwise OR to get all the bits
        // set except in the range from i to j
        int mask = left | right;

        // clear bits j through i
        int masked_n = n & mask;

        // move m into the correct position
        int m_shifted = m << i;

        // return the Bitwise OR of masked_n
        // and shifted_m
        return (masked_n | m_shifted);
    }

    // Driver Program to test above function
    public static void Main()
    {
        int n = 2, m = 4;
        int i = 2, j = 4;

        Console.WriteLine(setBits(n, m, i, j));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program for above implementation

// Function to set the bits
function setBits($n, $m, $i, $j)
{
    // number with all 1's
    $allOnes = ~0;

    // Set all the bits
    // in the left of j
    $left = $allOnes << ($j + 1);
```

```
    // Set all the bits
    // in the right of j
    $right = ((1 << $i) - 1);

    // Do Bitwsie OR to get all
    // the bits set except in
    // the range from i to j
    $mask = $left | $right;

    // clear bits j through i
    $masked_n = $n & $mask;

    // move m into the
    // correct position
    $m_shifted = $m << $i;

    // return the Bitwise OR
    // of masked_n and shifted_m
    return ($masked_n | $m_shifted);
}

// Driver Code
$n = 2; $m = 4;
$i = 2; $j = 4;
echo setBits($n, $m, $i, $j);

// This code is contributed by ajit
?>
```

Output :

```
18
```

Reference:
https://www.careercup.com/question?id=8863294

Improved By : jit_t

Source

https://www.geeksforgeeks.org/set-bits-n-equals-m-given-range/

Chapter 289

Set the K-th bit of a given number

Set the K-th bit of a given number - GeeksforGeeks

Given a number n and a value k. From the right, set the kth bit in the binary representation of n. The position of LSB(or last bit) is 0, second last bit is 1 and so on. Also, 0 <= k < x, where x is the number of bits in the binary representation of n.

Examples:

```
Input : n = 10, k = 2
Output : 14
(10)10 = (1010)2
Now, set the 2nd bit from right.
(14)10 = (1110)2
2nd bit has been set.

Input : n = 15, k = 3
Output : 15
3rd bit of 15 is already set.
```

To set any bit we use bitwise OR | operator. As we already know bitwise OR | operator evaluates each bit of the result to 1 if any of the operand's corresponding bit is set (1). In-order to set kth bit of a number we need to shift 1 k times to its left and then perform bitwise OR operation with the number and result of left shift performed just before.

```
In general, (1 << k) | n.
```

C++

```
 // C++ implementation to set the kth bit
// of the given number
#include <bits/stdc++.h>

using namespace std;

// function to set the kth bit
int setKthBit(int n, int k)
{
    // kth bit of n is being set by this operation
    return ((1 << k) | n);
}

// Driver program to test above
int main()
{
    int n = 10, k = 2;
    cout << "Kth bit set number = "
         << setKthBit(n, k);
    return 0;
}
```

Java

```
 // Java implementation to set the kth bit
// of the given number

class GFG {

// function to set the kth bit
static int setKthBit(int n, int k)
{
    // kth bit of n is being set by this operation
    return ((1 << k) | n);
}

// Driver code
public static void main(String arg[])
{
    int n = 10, k = 2;
    System.out.print("Kth bit set number = " +
                              setKthBit(n, k));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python implementation
# to set the kth bit
# of the given number

# function to set
# the kth bit
def setKthBit(n,k):

    # kth bit of n is being
    # set by this operation
    return ((1 << k) | n)

# Driver code

n = 10
k = 2

print("Kth bit set number = ",
          setKthBit(n, k))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# implementation to set the
// kth bit of the given number
using System;

class GFG {

// function to set the kth bit
static int setKthBit(int n, int k)
{
    // kth bit of n is being set
    // by this operation
    return ((1 << k) | n);
}

// Driver code
public static void Main()
{
    int n = 10, k = 2;
    Console.Write("Kth bit set number = "
                    + setKthBit(n, k));

}
}
```

```
// This code is contributed by
// Smitha Dinesh Semwal.
```

PHP

```
 <?php
// PHP implementation to
// set the kth bit of
// the given number

// function to set
// the kth bit
function setKthBit($n, $k)
{
    // kth bit of n is being
    // set by this operation
    return ((1 << $k) | $n);
}

// Driver Code
$n = 10; $k = 2;
echo "Kth bit set number = ",
           setKthBit($n, $k);

// This code is contributed by m_kit
?>
```

Output:

```
Kth bit set number = 14
```

Improved By : Smitha Dinesh Semwal, jit_t

Source

https://www.geeksforgeeks.org/set-k-th-bit-given-number/

Chapter 290

Set the Left most unset bit

Set the Left most unset bit - GeeksforGeeks

Given an integer, set the leftmost unset bit. Leftmost unset bit is the first unset bit after most significant set bit. If all bits (after most significant set bit) are set, then return the number.

Examples:

```
Input : 10
Output : 14
10 = 1 0 1 0    // 10 binary
14 = 1 1 1 0    // after set left most unset bit

Input : 15
Output : 15
15 = 1 1 1 1    // 15 binary
15 = 1 1 1 1    // because all bits are set
```

Approach:-
1. Return the number if all bits are set.
2. Traverse all the bit to get the last unset bit.
3. Take OR with the original number and the unset bit.

Below is the implementation of the approach.

C++

```
 // CPP program to set the leftmost unset bit
#include <iostream>
using namespace std;
```

```
// set left most unset bit
int setleftmostunsetbit(int n)
{
    // if number contain all
    // 1 then return n
    if ((n & (n + 1)) == 0)
        return n;

    // Find position of leftmost unset bit.
    int pos = 0;
    for (int temp=n, count=0; temp>0;
                        temp>>=1, count++)

        // if temp L.S.B is zero
        // then unset bit pos is
        // change
        if ((temp & 1) == 0)
            pos = count;

    // return OR of number and
    // unset bit pos
    return (n | (1 << (pos)));
}

// Driver Function
int main()
{
    int n = 10;
    cout << setleftmostunsetbit(n);
    return 0;
}
```

Java

```
 // Java program to set
// the leftmost unset bit
import java.io.*;

class GFG
{
    // set left most unset bit
    static int setleftmostunsetbit(int n)
    {
        // if number contain all
        // 1 then return n
        if ((n & (n + 1)) == 0)
            return n;
```

```
        // Find position of leftmost unset bit.
        int pos = 0;
        for (int temp = n, count = 0; temp > 0;
                            temp >>= 1, count++)

            // if temp L.S.B is zero
            // then unset bit pos is
            // change
            if ((temp & 1) == 0)
                pos = count;

        // return OR of number and
        // unset bit pos
        return (n | (1 << (pos)));
    }

    // Driver Function
    public static void main (String[] args)
    {
        int n = 10;
        System.out.println(setleftmostunsetbit(n));
    }
}

// This code is contributed by Ansu Kumari
```

Python3

```
 # Python program to set the leftmost unset bit

# Set left most unset bit
def setleftmostunsetbit(n):
    # if number contain all
    # 1 then return n
    if not (n & (n + 1)):
        return n

    # Find position of leftmost unset bit
    pos, temp, count = 0, n, 0

    while temp:
        # if temp L.S.B is zero
        # then unset bit pos is
        # change
        if not (temp & 1):
            pos = count

        count += 1; temp>>=1
```

```
    # return OR of number and
    # unset bit pos
    return (n | (1 << (pos)))


# Driver Function
n = 10
print(setleftmostunsetbit(n))

# This code is contributed by Ansu Kumari
```

C#

```
 // C# program to set
// the leftmost unset bit
using System;

class GFG
{
    // set left most unset bit
    static int setleftmostunsetbit(int n)
    {
        // if number contain all
        // 1 then return n
        if ((n & (n + 1)) == 0)
            return n;

        // Find position of leftmost unset bit.
        int pos = 0;
        for (int temp = n, count = 0; temp > 0;
                            temp >>= 1, count++)

            // if temp L.S.B is zero
            // then unset bit pos is
            // change
            if ((temp & 1) == 0)
                pos = count;

        // return OR of number and
        // unset bit pos
        return (n | (1 << (pos)));
    }

    // Driver Function
    public static void Main ()
    {
        int n = 10;
```

```
        Console.WriteLine(setleftmostunsetbit(n));
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// php program to set the
// leftmost unset bit

// set left most unset bit
function setleftmostunsetbit($n)
{

    // if number contain all
    // 1 then return n
    if (($n & ($n + 1)) == 0)
        return $n;

    // Find position of leftmost
    // unset bit.
    $pos = 0;
    for ($temp = $n, $count = 0; $temp > 0;
                     $temp >>= 1, $count++)

        // if temp L.S.B is zero
        // then unset bit pos is
        // change
        if (($temp & 1) == 0)
            $pos = $count;

    // return OR of number
    // and unset bit pos
    return ($n | (1 << ($pos)));
}

    // Driver code
    $n = 10;
    echo setleftmostunsetbit($n);

//This code is contributed by mits
?>
```

Output:

```
14
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/set-left-unset-bit/

Chapter 291

Set the rightmost unset bit

Set the rightmost unset bit - GeeksforGeeks

Given a non-negative number **n**. The problem is to set the rightmost unset bit in the binary representation of **n**. If there are no unset bits, then just leave the number as it is.

Examples:

```
Input : 21
Output : 23
(21)10 = (10101)2
Rightmost unset bit is at position 2(from right) as
highlighted in the binary representation of 21.
(23)10 = (10111)2
The bit at position 2 has been set.

Input : 15
Output : 15
```

Approach: Following are the steps:

1. If **n** = 0, return 1.
2. If all bits of **n** are set, return n. Refer this post.
3. Else perform bitwise not on the given number(operation equivalent to 1's complement). Let it be **num** = ~n.
4. Get the position of rightmost set bit of **num**. Let the position be **pos**.
5. Return **(1 << (pos − 1)) | n**.

C++

```
// C++ implementation to set the rightmost unset bit
```

```
#include <bits/stdc++.h>
using namespace std;

// function to find the position
// of rightmost set bit
int getPosOfRightmostSetBit(int n)
{
    return log2(n&-n)+1;
}

int setRightmostUnsetBit(int n)
{
    // if n = 0, return 1
    if (n == 0)
        return 1;

    // if all bits of 'n' are set
    if ((n & (n + 1)) == 0)
        return n;

    // position of rightmost unset bit in 'n'
    // passing ~n as argument
    int pos = getPosOfRightmostSetBit(~n);

    // set the bit at position 'pos'
    return ((1 << (pos - 1)) | n);
}

// Driver program to test above
int main()
{
    int n = 21;
    cout << setRightmostUnsetBit(n);
    return 0;
}
```

Java

```
 // Java implementation to set
// the rightmost unset bit

class GFG {

// function to find the position
// of rightmost set bit
static int getPosOfRightmostSetBit(int n)
{
    return (int)((Math.log10(n & -n)) / (Math.log10(2))) + 1;
```

```
}

static int setRightmostUnsetBit(int n)
{
    // if n = 0, return 1
    if (n == 0)
    return 1;

    // if all bits of 'n' are set
    if ((n & (n + 1)) == 0)
    return n;

    // position of rightmost unset bit in 'n'
    // passing ~n as argument
    int pos = getPosOfRightmostSetBit(~n);

    // set the bit at position 'pos'
    return ((1 << (pos - 1)) | n);
}

// Driver code
public static void main(String arg[]) {
    int n = 21;
    System.out.print(setRightmostUnsetBit(n));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python implementation to
# set the rightmost unset bit
import math

# function to find the position
# of rightmost set bit
def getPosOfRightmostSetBit(n):

    return int(math.log2(n&-n)+1)


def setRightmostUnsetBit(n):

    # if n = 0, return 1
    if (n == 0):
        return 1
```

```
    # if all bits of 'n' are set
    if ((n & (n + 1)) == 0):
        return n

    # position of rightmost unset bit in 'n'
    # passing ~n as argument
    pos = getPosOfRightmostSetBit(~n)

    # set the bit at position 'pos'
    return ((1 << (pos - 1)) | n)

# Driver code

n = 21
print(setRightmostUnsetBit(n))

# This code is contributed
# by Anant Agarwal.
```

Output:

```
23
```

Source

https://www.geeksforgeeks.org/set-rightmost-unset-bit/

Chapter 292

Shuffle a pack of cards and answer the query

Shuffle a pack of cards and answer the query - GeeksforGeeks

Given a pack of 2^N cards (0 … 2^N – 1), shuffle it in N steps. At step k ($0 < k < N$) we divide the deck into 2k equal-sized decks. Each one of those decks is reordered by having all the cards that lie on even positions first, followed by all cards that lie on odd positions (the order is preserved in each one of the two subsequences). Now, we are given a key (index). We have to answer the card on that position (0-based indexing). Examples:

```
Input : N = 3 (Size = 2^N), Key = 3
Output : 6
Explanation :
Pack :      0 1 2 3 4 5 6 7
Shuffle 1 : 0 2 4 6|1 3 5 7
Shuffle 2 : 0 4|2 6|1 5|3 7
Card at index 3 : 6
```

Method 1 : We can simply simulate the whole process and find the exact order of the cards after all the N shuffles are done.
Time Complexity : O(N * 2^N)

Method 2 :
Let us try to find the binary representation of Key and the final answer and try to spot some observations based on it.

Let N = 3

Below is the table :

Key ANS
000 000

001 100
010 010
011 110
100 001
101 101
110 011
111 111

It is clearly visible that the answer is the reverse of binary representation of Key.

C++

```
 // C++ program to find the card at given index
// after N shuffles
#include <bits/stdc++.h>
using namespace std;

// function to find card at given index
void shuffle(int N, int key)
{

    // Answer will be reversal of N bits from MSB
    unsigned int NO_OF_BITS = N;
    unsigned int reverse_num = 0, temp;

    // Calculating the reverse binary representation
    for (int i = 0; i < NO_OF_BITS; i++) {
        temp = (key & (1 << i));
        if (temp)
            reverse_num |= (1 << ((NO_OF_BITS - 1) - i));
    }

    // Printing the result
    cout << reverse_num;
}

// driver code
int main()
{
    // No. of Shuffle Steps
    int N = 3;

    // Key position
    unsigned int key = 3;

    shuffle(N, key);
    return 0;
}
```

Java

```
 // Java program to find the card at given index
// after N shuffles
class GFG {

    // function to find card at given index
    static void shuffle(int N, int key)
    {

        // Answer will be reversal of N bits from MSB
        int NO_OF_BITS = N;
        int reverse_num = 0, temp;

        // Calculating the reverse binary representation
        for (int i = 0; i < NO_OF_BITS; i++) {
            temp = (key & (1 << i));
            if (temp>0)
                reverse_num |= (1 << ((NO_OF_BITS - 1) - i));
        }

        // Printing the result
        System.out.print(reverse_num);
    }

    //Driver code
    public static void main (String[] args)
    {

        // No. of Shuffle Steps
        int N = 3;

        // Key position
        int key = 3;

        shuffle(N, key);
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find the card
# at given index after N shuffles

# Function to find card at given index
```

```
def shuffle(N, key):

    # Answer will be reversal
    # of N bits from MSB
    NO_OF_BITS = N
    reverse_num = 0

    # Calculating the reverse binary representation
    for i in range(NO_OF_BITS):
        temp = (key & (1 << i))
        if (temp):
            reverse_num |= (1 << ((NO_OF_BITS - 1) - i))

    # Printing the result
    print(reverse_num)

# Driver code

# No. of Shuffle Steps
N = 3

# Key position
key = 3
shuffle(N, key)

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to find the card at given index
// after N shuffles
using System;

class GFG {

    // function to find card at given index
    static void shuffle(int N, int key)
    {

        // Answer will be reversal of N bits from MSB
        int NO_OF_BITS = N;
        int reverse_num = 0, temp;

        // Calculating the reverse binary representation
        for (int i = 0; i < NO_OF_BITS; i++) {
            temp = (key & (1 << i));
            if (temp > 0)
                reverse_num |= (1 << ((NO_OF_BITS - 1) - i));
```

```
        }

        // Printing the result
        Console.Write(reverse_num);
    }

    //Driver code
    public static void Main()
    {

        // No. of Shuffle Steps
        int N = 3;

        // Key position
        int key = 3;

        shuffle(N, key);
    }
}

// This code is contributed by Anant Agarwal.
```

Output:

```
6
```

Source

https://www.geeksforgeeks.org/shuffle-pack-cards-answer-query/

Chapter 293

Smallest number whose set bits are maximum in a given range

Smallest number whose set bits are maximum in a given range - GeeksforGeeks

Given a positive integer '**l**' and '**r**'. Find the smallest number '**n**' such that **l <= n <= r** and count of number of set bits(number of '1's in binary representation) is maximum as possible.
Examples :

```
Input: 1 4
Output: 3
Explanation:
Binary representation from '1' to '4':
110 = 0012
210 = 0102
310 = 0112
110 = 1002
Thus number '3' has maximum set bits = 2

Input: 1 10
Output: 7
```

Simple approach is to traverse from 'l' to 'r' and count the set bits for each 'x'(l <= n <= r) and print the number whose count is maximum among them. Time complexity of this approach is O(n*log(r)).

C++

```
// C++ program to find number whose set
```

```
// bits are maximum among 'l' and 'r'
#include <bits/stdc++.h>
using namespace std;

// Returns smallest number whose set bits
// are maximum in given range.
int countMaxSetBits(int left, int right)
{
    // Initialize the maximum count and
    // final answer as 'num'
    int max_count = -1, num;
    for (int i = left; i <= right; ++i) {
        int temp = i, cnt = 0;

        // Traverse for every bit of 'i'
        // number
        while (temp) {
            if (temp & 1)
                ++cnt;
            temp >>= 1;
        }

        // If count is greater than previous
        // calculated max_count, update it
        if (cnt > max_count) {
            max_count = cnt;
            num = i;
        }
    }
    return num;
}

// Driver code
int main()
{
    int l = 1, r = 5;
    cout << countMaxSetBits(l, r) << "\n";

    l = 1, r = 10;
    cout << countMaxSetBits(l, r);
    return 0;
}
```

Python 3

```
 # Python code to find number whose set
# bits are maximum among 'l' and 'r'
```

```
def countMaxSetBits( left, right):
    max_count = -1
    for i in range(left, right+1):
        temp = i
        cnt = 0

        # Traverse for every bit of 'i'
        # number
        while temp:
            if temp & 1:
                cnt +=1
            temp = temp >> 1

        # If count is greater than previous
        # calculated max_count, update it
        if cnt > max_count:
            max_count = cnt
            num=i
    return num

# driver code
l = 1
r = 5
print(countMaxSetBits(l, r))
l = 1
r = 10
print(countMaxSetBits(l, r))

# This code is contributed by "Abhishek Sharma 44"
```

PHP

```
 <?php
// PHP program to find number
// whose set bits are maximum
// among 'l' and 'r'

// Returns smallest number
// whose set bits are maximum
// in given range.

function countMaxSetBits($left, $right)
{
    // Initialize the maximum
    // count and final answer
    // as 'num'
    $max_count = -1; $num;
    for ($i = $left; $i <= $right; ++$i)
```

```
    {
        $temp = $i; $cnt = 0;

        // Traverse for every
        // bit of 'i' number
        while ($temp)
        {
            if ($temp & 1)
                ++$cnt;
            $temp >>= 1;
        }

        // If count is greater than
        // previous calculated
        // max_count, update it
        if ($cnt > $max_count)
        {
            $max_count = $cnt;
            $num = $i;
        }
    }
    return $num;
}

// Driver code
$l = 1; $r = 5;
echo countMaxSetBits($l, $r), "\n";

$l = 1; $r = 10;
echo countMaxSetBits($l, $r);

// This code is contributed by m_kit
?>
```

Output :

```
3
7
```

Efficient approach is to use bit-manipulation. Instead of iterating for every number from 'l' to 'r', iterate only after updating the desired number('num') i.e., take the bitwise 'OR' of number with the consecutive number. For instance,

```
Let l = 2, and r = 10
1. num = 2
2. x = num OR (num + 1)
```

```
       = 2 | 3 = 010 | 011 = 011
   num = 3(011)
3. x = 3 | 4 = 011 | 100 = 111
   num = 7(111)
4. x = 7 | 8 = 0111 | 1000 = 1111
   Since 15(11112) is greater than
   10, thus stop traversing for next number.
5. Final answer = 7
```

C++

```
 // C++ program to find number whose set
// bits are maximum among 'l' and 'r'
#include <bits/stdc++.h>
using namespace std;

// Returns smallest number whose set bits
// are maximum in given range.
int countMaxSetBits(int left, int right)
{
    while ((left | (left + 1)) <= right)
        left |= left + 1;

    return left;
}

// Driver code
int main()
{
    int l = 1, r = 5;
    cout << countMaxSetBits(l, r) << "\n";

    l = 1, r = 10;
    cout << countMaxSetBits(l, r) ;
    return 0;
}
```

Java

```
 // Java program to find number
// whose set bits are maximum
// among 'l' and 'r'
import java.io.*;

class GFG
{
```

```
    // Returns smallest number
    // whose set bits are
    // maximum in given range.
    static int countMaxSetBits(int left,
                               int right)
    {
    while ((left | (left + 1)) <= right)
        left |= left + 1;

    return left;
    }

// Driver code
public static void main (String[] args)
{
    int l = 1;
    int r = 5;
    System.out.println(countMaxSetBits(l, r));

    l = 1;
    r = 10;
    System.out.println(countMaxSetBits(l, r));
}
}

// This code is contributed by @ajit
```

Python3

```
 # Python code to find number whose set
# bits are maximum among 'l' and 'r'

def countMaxSetBits( left, right):

    while(left | (left+1)) <= right:
        left |= left+1
    return left

# driver code
l = 1
r = 5
print(countMaxSetBits(l, r))
l = 1
r = 10
print(countMaxSetBits(l, r))

# This code is contributed by "Abhishek Sharma 44"
```

C#

```
 // C# program to find number
// whose set bits are maximum
// among 'l' and 'r'
using System;

class GFG
{

    // Returns smallest number
    // whose set bits are
    // maximum in given range.
    static int countMaxSetBits(int left,
                               int right)
    {
    while ((left | (left + 1)) <= right)
        left |= left + 1;

    return left;
    }

// Driver code
static public void Main ()
{
    int l = 1;
    int r = 5;
    Console.WriteLine(countMaxSetBits(l, r));

    l = 1;
    r = 10;
    Console.WriteLine(countMaxSetBits(l, r));
}
}

// This code is contributed by @ajit
```

PHP

```
 <?php
// PHP program to find number
// whose set bits are maximum
// among 'l' and 'r'

// Returns smallest number
// whose set bits are
// maximum in given range.
```

```
function countMaxSetBits($left,
                         $right)
{
    while (($left | ($left + 1)) <= $right)
        $left |= $left + 1;

    return $left;
}

// Driver code
$l = 1 ; $r = 5;
echo countMaxSetBits($l, $r) , "\n";

$l = 1; $r = 10;
echo countMaxSetBits($l, $r) ;

// This code is contributed by aj_36
?>
```

Output :

```
3
7
```

Time complexity: O(log(n))
Auxiliary space: O(1)

Improved By : jit_t

Source

https://www.geeksforgeeks.org/smallest-number-whose-set-bits-maximum-given-range/

Chapter 294

Smallest of three integers without comparison operators

Smallest of three integers without comparison operators - GeeksforGeeks

Write a program to find the smallest of three integers, without using any of the comparison operators.

Let 3 input numbers be x, y and z.

Method 1 (Repeated Subtraction)
Take a counter variable c and initialize it with 0. In a loop, repeatedly subtract x, y and z by 1 and increment c. The number which becomes 0 first is the smallest. After the loop terminates, c will hold the minimum of 3.

C

```
 #include<stdio.h>

int smallest(int x, int y, int z)
{
  int c = 0;
  while ( x && y && z )
  {
      x--;  y--; z--; c++;
  }
  return c;
}

int main()
{
   int x = 12, y = 15, z = 5;
   printf("Minimum of 3 numbers is %d", smallest(x, y, z));
```

```
   return 0;
}
```

Java

```
 class GFG{

    static int smallest(int x, int y, int z)
    {
        int c = 0;

        while ( x != 0 && y != 0 && z != 0 )
        {
            x--; y--; z--; c++;
        }

        return c;
    }

    public static void main(String[] args)
    {
        int x = 12, y = 15, z = 5;

        System.out.printf("Minimum of 3" +
              " numbers is %d", smallest(x, y, z));
    }
}

// This code is contributed by  Smitha Dinesh Semwal.
```

Python3

```
 # Python3 program to find Smallest
# of three integers without
# comparison operators

def smallest(x, y, z):
    c = 0

    while ( x and y and z ):
        x=x-1
        y=y-1
        z=z-1
        c=c+1

    return c
```

```
# Driver Code
x = 12
y = 15
z = 5
print("Minimum of 3 numbers is",
       smallest(x, y, z))

# This code is contributed by Anshika Goyal
```

C#

```
 // C# program to find Smallest of three
// integers without comparison operators
using System;

class GFG
{
    static int smallest(int x, int y, int z)
    {
        int c = 0;

        while ( x != 0 && y != 0 && z != 0 )
        {
            x--; y--; z--; c++;
        }

        return c;
    }

    // Driver Code
    public static void Main()
    {
        int x = 12, y = 15, z = 5;

        Console.Write("Minimum of 3" +
            " numbers is " + smallest(x, y, z));
    }
}

// This code is contributed by Sam007
```

php

```
 <?php
// php program to find Smallest
// of three integers without
// comparison operators
```

```
function smallest($x, $y, $z)
{
    $c = 0;
    while ( $x && $y && $z )
    {
        $x--; $y--; $z--; $c++;
    }

    return $c;
}

// Drive code
$x = 12;
$y = 15;
$z = 5;
echo "Minimum of 3 numbers is ".
             smallest($x, $y, $z);

// This code is contributed by Sam007
?>
```

This methid doesn't work for negative numbers. Method 2 works for negative nnumbers also.

Method 2 (Use Bit Operations)
Use method 2 of this post to find minimum of two numbers (We can't use Method 1 as Method 1 uses comparison operator). Once we have functionality to find minimum of 2 numbers, we can use this to find minimum of 3 numbers.

```
 // See mthod 2 of https://www.geeksforgeeks.org/compute-the-minimum-or-maximum-max-of-two-intege
#include<stdio.h>
#define CHAR_BIT 8

/*Function to find minimum of x and y*/
int min(int x, int y)
{
  return  y + ((x - y) & ((x - y) >>
            (sizeof(int) * CHAR_BIT - 1)));
}

/* Function to find minimum of 3 numbers x, y and z*/
int smallest(int x, int y, int z)
{
    return min(x, min(y, z));
}

int main()
{
```

```
   int x = 12, y = 15, z = 5;
   printf("Minimum of 3 numbers is %d", smallest(x, y, z));
   return 0;
}
```

Method 3 (Use Division operator)
We can also use division operator to find minimum of two numbers. If value of (a/b) is zero, then b is greater than a, else a is greater. Thanks to gopinathand Vigneshfor suggesting this method.

```
 #include <stdio.h>

// Using division operator to find minimum of three numbers
int smallest(int x, int y, int z)
{
    if (!(y/x))  // Same as "if (y < x)"
        return (!(y/z))? y : z;
    return (!(x/z))? x : z;
}

int main()
{
    int x = 78, y = 88, z = 68;
    printf("Minimum of 3 numbers is %d", smallest(x, y, z));
    return 0;
}
```

Improved By : Sam007

Source

https://www.geeksforgeeks.org/smallest-of-three-integers-without-comparison-operators/

Chapter 295

Smallest perfect power of 2 greater than n (without using arithmetic operators)

Smallest perfect power of 2 greater than n (without using arithmetic operators) - GeeksforGeeks

Given a non-negative integer **n**. The problem is to find the smallest perfect power of 2 which is greater than **n** without using the arithmetic operators.

Examples :

```
Input : n = 10
Output : 16

Input : n = 128
Output : 256
```

Algorithm :

```
perfectPowerOf2(n)
    Initialize per_pow = 1
    while n > 0
        per_pow = per_pow << 1
        n = n >> 1
    return per_pow
```

C++

```
// C++ implementation of smallest perfect power
```

```
// of 2 greater than n
#include <bits/stdc++.h>

using namespace std;

// Function to find smallest perfect power
// of 2 greater than n
unsigned int perfectPowerOf2(unsigned int n)
{
    // To store perfect power of 2
    unsigned int per_pow = 1;

    while (n > 0)
    {
        // bitwise left shift by 1
        per_pow = per_pow << 1;

        // bitwise right shift by 1
        n = n >> 1;
    }

    // Required perfect power of 2
    return per_pow;
}

// Driver program to test above
int main()
{
    unsigned int n = 128;
    cout << "Perfect power of 2 greater than "
        << n << ": " << perfectPowerOf2(n);
    return 0;
}
```

Java

```
 // JAVA Code for Smallest perfect
// power of 2 greater than n
import java.util.*;

class GFG {

    // Function to find smallest perfect
    // power of 2 greater than n
    static int perfectPowerOf2( int n)
    {
        // To store perfect power of 2
         int per_pow = 1;
```

```
        while (n > 0)
        {
            // bitwise left shift by 1
            per_pow = per_pow << 1;

            n = n >> 1;
        }

        // Required perfect power of 2
        return per_pow;
    }

    // Driver program
    public static void main(String[] args)
    {
         int n = 12;
         System.out.println("Perfect power of 2 greater than "
                            + n + ": " + perfectPowerOf2(n));
        }
    }

    //This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python3 implementation of smallest
# perfect power of 2 greater than n

# Function to find smallest perfect
# power of 2 greater than n
def perfectPowerOf2( n ):

    # To store perfect power of 2
    per_pow = 1

    while n > 0:

        # bitwise left shift by 1
        per_pow = per_pow << 1

        # bitwise right shift by 1
        n = n >> 1

    # Required perfect power of 2
    return per_pow

# Driver program to test above
```

```
n = 128
print("Perfect power of 2 greater than",
           n, ":",perfectPowerOf2(n))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# Code for Smallest perfect
// power of 2 greater than n
using System;

class GFG {

    // Function to find smallest perfect
    // power of 2 greater than n
    static int perfectPowerOf2(int n)
    {
        // To store perfect power of 2
        int per_pow = 1;

        while (n > 0)
        {
            // bitwise left shift by 1
            per_pow = per_pow << 1;

            n = n >> 1;
        }

        // Required perfect power of 2
        return per_pow;
    }

    // Driver program
    public static void Main()
    {
        int n = 128;
        Console.WriteLine("Perfect power of 2 greater than " +
                          n + ": " + perfectPowerOf2(n));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
```

```
// php implementation of
// smallest perfect power
// of 2 greater than n

// Function to find smallest
// perfect power of 2
// greater than n
function perfectPowerOf2($n)
{

    // To store perfect power of 2
    $per_pow = 1;

    while ($n > 0)
    {
        // bitwise left shift by 1
        $per_pow = $per_pow << 1;

        // bitwise right shift by 1
        $n = $n >> 1;
    }

    // Required perfect power of 2
    return $per_pow;
}

    // Driver code
    $n = 128;
    echo "Perfect power of 2 greater than ".
          $n . ": ".perfectPowerOf2($n);

// This code is contributed by mits
?>
```

Output:

```
Perfect power of 2 greater than 128: 256
```

Time Complexity: **O(num)**, where **num** is the number of bits in the binary representation of **n**.

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/smallest-perfect-power-2-greater-n-without-using-arithmetic-operators/

Chapter 296

Smallest power of 2 greater than or equal to n

Smallest power of 2 greater than or equal to n - GeeksforGeeks

Write a function that, for a given no n, finds a number p which is greater than or equal to n and is a smallest power of 2.
Examples :

```
Input : n = 5
Output: 8

Input  : n = 17
Output : 32

Input  : n = 32
Output : 32
```

There are plenty of solutions for this. Let us take the example of 17 to explain some of them.

Method 1(Using Log of the number)

```
1.  Calculate Position of set bit in p(next power of 2):
    pos =  ceil(lgn)  (ceiling of log n with base 2)
2.  Now calculate p:
    p   = pow(2, pos)
```

Example :

```
Let us try for 17
        pos = 5
        p   = 32
```

Method 2 (By getting the position of only set bit in result)

```
/* If n is a power of 2 then return n */
1  If (n & !(n&(n-1))) then return n
2  Else keep right shifting n until it becomes zero
    and count no of shifts
    a. Initialize: count = 0
    b. While n ! = 0
            n = n>>1
            count = count + 1

 /* Now count has the position of set bit in result */
3  Return (1 << count)
```

Example :

```
Let us try for 17
             count = 5
             p     = 32
```

C

```
 #include<stdio.h>

unsigned int nextPowerOf2(unsigned int n)
{
unsigned count = 0;

// First n in the below condition
// is for the case where n is 0
if (n && !(n & (n - 1)))
    return n;

while( n != 0)
{
    n >>= 1;
    count += 1;
}
```

```
return 1 << count;
}

// Driver Code
int main()
{
unsigned int n = 0;
printf("%d", nextPowerOf2(n));
return 0;
}
```

Java

```
 // Java program to find
// smallest power of 2
// greater than or equal to n
import java.io.*;

class GFG
{
    static int nextPowerOf2(int n)
    {
        int count = 0;

        // First n in the below
        // condition is for the
        // case where n is 0
        if (n > 0 && (n & (n - 1)) == 0)
            return n;

        while(n != 0)
        {
            n >>= 1;
            count += 1;
        }

        return 1 << count;
    }

    // Driver Code
    public static void main(String args[])
    {
        int n = 0;
        System.out.println(nextPowerOf2(n));
    }
}
```

```
// This article is contributed
// by Anshika Goyal.
```

Python3

```
 def nextPowerOf2(n):
    count = 0;

    # First n in the below
    # condition is for the
    # case where n is 0
    if (n and not(n & (n - 1))):
        return n

    while( n != 0):
        n >>= 1
        count += 1

    return 1 << count;


# Driver Code
n = 0
print(nextPowerOf2(n))
# This code is contributed
# by Smitha Dinesh Semwal
```

C#

```
 // C# program to find smallest
// power of 2 greater than
// or equal to n
using System;

class GFG
{
    static int nextPowerOf2(int n)
    {
        int count = 0;

        // First n in the below
        // condition is for the
        // case where n is 0
        if (n > 0 && (n & (n - 1)) == 0)
            return n;

        while(n != 0)
```

```
        {
            n >>= 1;
            count += 1;
        }

        return 1 << count;
    }

    // Driver Code
    public static void Main()
    {
        int n = 0;
        Console.WriteLine(nextPowerOf2(n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to find smallest
// power of 2 greater than or
// equal to n

function nextPowerOf2($n)
{
$count = 0;

// First n in the below condition
// is for the case where n is 0
if ($n && !($n&($n - 1)))
    return $n;

while($n != 0)
{
    $n >>= 1;
    $count += 1;
}

return 1 << $count;
}

// Driver Code
$n = 0;
echo (nextPowerOf2($n));

// This code is contributed by vt_m
```

```
?>
```

Output :

```
1
```

Method 3(Shift result one by one)
Thanks to coderyogi for suggesting this method . This method is a variation of method 2 where instead of getting count, we shift the result one by one in a loop.

C

```
 #include<stdio.h>
unsigned int nextPowerOf2(unsigned int n)
{
    unsigned int p = 1;
    if (n && !(n & (n - 1)))
        return n;

    while (p < n)
        p <<= 1;

    return p;
}

// Driver Code
int main()
{
unsigned int n = 5;
printf("%d", nextPowerOf2(n));
return 0;
}
```

Java

```
 // Java program to find smallest
// power of 2 greater than or
// equal to n
import java.io.*;

class GFG
{
    static int nextPowerOf2(int n)
    {
```

```
        int p = 1;
        if (n > 0 && (n & (n - 1)) == 0)
            return n;

        while (p < n)
            p <<= 1;

        return p;
    }

    // Driver Code
    public static void main(String args[])
    {
        int n = 5;
        System.out.println(nextPowerOf2(n));
    }
}

// This article is contributed
// by Anshika Goyal.
```

Python3

```
 def nextPowerOf2(n):

    p = 1
    if (n and not(n & (n - 1))):
        return n

    while (p < n) :
        p <<= 1

    return p;


# Driver Code
n = 5
print(nextPowerOf2(n));

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to find smallest
// power of 2 greater than or
// equal to n
```

```
using System;

class GFG
{

    static int nextPowerOf2(int n)
    {
        int p = 1;
        if (n > 0 && (n & (n - 1)) == 0)
            return n;

        while (p < n)
            p <<= 1;

        return p;
    }

    // Driver Code
    public static void Main()
    {
        int n = 5;
        Console.Write(nextPowerOf2(n));
    }
}

// This code is contributed by Smitha.
```

PHP

```
 <?php

function nextPowerOf2($n)
{
    $p = 1;
    if ($n && !($n & ($n - 1)))
        return $n;

    while ($p < $n)
        $p <<= 1;

    return $p;
}

// Driver Code
$n = 5;
echo ( nextPowerOf2($n));

// This code is contributed by vt_m.
```

```
?>
```

Output :

```
8
```

Time Complexity: O(lgn)

Method 4(Customized and Fast)

```
    1. Subtract n by 1
       n = n -1

    2. Set all bits after the leftmost set bit.

    /* Below solution works only if integer is 32 bits */
                n = n | (n >> 1);
                n = n | (n >> 2);
                n = n | (n >> 4);
                n = n | (n >> 8);
                n = n | (n >> 16);
    3. Return n + 1
```

Example :

```
Steps 1 & 3 of above algorithm are to handle cases
of power of 2 numbers e.g., 1, 2, 4, 8, 16,

    Let us try for 17(10001)
    step 1
       n = n - 1 = 16 (10000)
    step 2
       n = n | n >> 1
       n = 10000 | 01000
       n = 11000
       n = n | n >> 2
       n = 11000 | 00110
       n = 11110
       n = n | n >> 4
       n = 11110 | 00001
       n = 11111
       n = n | n >> 8
       n = 11111 | 00000
```

```
       n = 11111
       n = n | n >> 16
       n = 11110 | 00000
       n = 11111

    step 3: Return n+1
     We get n + 1 as 100000 (32)
```

Program:

C

```
 #include <stdio.h>
// Finds next power of two
// for n. If n itself is a
// power of two then returns n
unsigned int nextPowerOf2(unsigned int n)
{
    n--;
    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;
    n++;
    return n;
}

// Driver Code
int main()
{
    unsigned int n = 5;
    printf("%d", nextPowerOf2(n));
    return 0;
}
```

Java

```
 // Java program to find smallest
// power of 2 greater than or
// equal to n
import java.io.*;

class GFG
{
    // Finds next power of two
    // for n. If n itself is a
```

```
    // power of two then returns n
    static int nextPowerOf2(int n)
    {
        n--;
        n |= n >> 1;
        n |= n >> 2;
        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;
        n++;

        return n;
    }

    // Driver Code
    public static void main(String args[])
    {
        int n = 5;
        System.out.println(nextPowerOf2(n));
    }
}

// This article is contributed
// by Anshika Goyal.
```

Python 3

```
 # Finds next power of two
# for n. If n itself is a
# power of two then returns n
def nextPowerOf2(n):

    n -= 1
    n |= n >> 1
    n |= n >> 2
    n |= n >> 4
    n |= n >> 8
    n |= n >> 16
    n += 1
    return n

# Driver program to test
# above function
n = 5
print(nextPowerOf2(n))

# This code is contributed
# by Smitha
```

C#

```
 // C# program to find smallest
// power of 2 greater than or
// equal to n
using System;

class GFG
{

    // Finds next power of two
    // for n. If n itself is a
    // power of two then returns n
    static int nextPowerOf2(int n)
    {
        n--;
        n |= n >> 1;
        n |= n >> 2;
        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;
        n++;

        return n;
    }

    // Driver Code
    public static void Main()
    {
        int n = 5;
        Console.WriteLine(nextPowerOf2(n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to find smallest
// power of 2 greater than or
// equal to n

// Finds next power of
// two for n. If n itself
// is a power of two then
// returns n
```

```
function nextPowerOf2($n)
{
    $n--;
    $n |= $n >> 1;
    $n |= $n >> 2;
    $n |= $n >> 4;
    $n |= $n >> 8;
    $n |= $n >> 16;
    $n++;
    return $n;
}

    // Driver Code
    $n = 5;
    echo nextPowerOf2($n);

// This code contributed by Ajit
?>
```

Output :

```
8
```

Time Complexity : O(lgn)

Related Post :
Highest power of 2 less than or equal to given number

References :
http://en.wikipedia.org/wiki/Power_of_2

Improved By : vt_m, jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/smallest-power-of-2-greater-than-or-equal-to-n/

Chapter 297

Space optimization using bit manipulations

Space optimization using bit manipulations - GeeksforGeeks

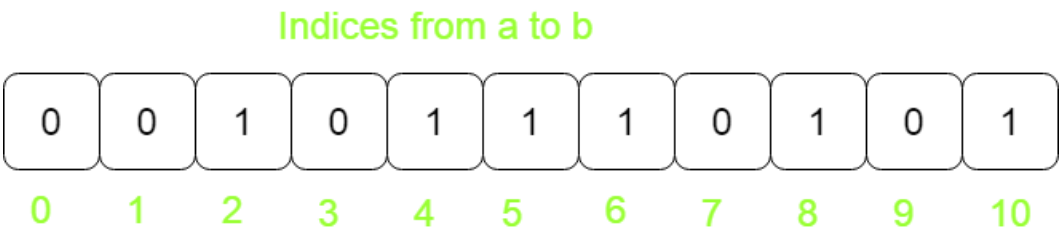
There are many situations where we use integer values as index in array to see presence or absence, we can use bit manipulations to optimize space in such problems.

Let us consider below problem as an example.

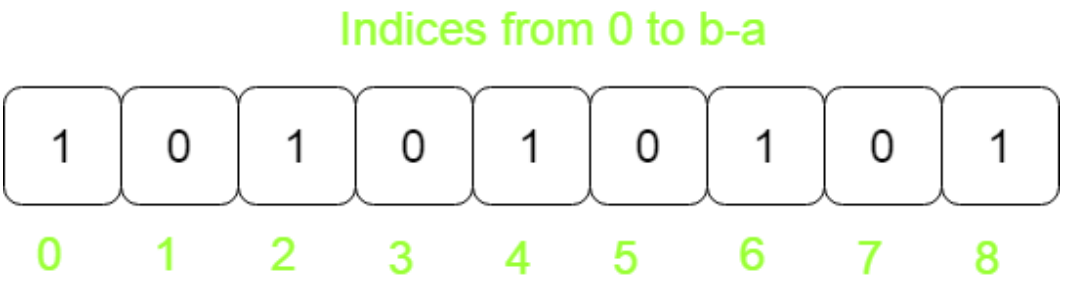
Given two numbers say a and b, mark the multiples of 2 and 5 between a and b using less than O(|b – a|) space and output each of the multiples.
Note : We have to **mark** the multiples i.e save (key, value) pairs in memory such that each key either have value as 1 or 0 representing as multiple of 2 or 5 or not respectively.
Examples :

```
Input : 2 10
Output : 2 4 5 6 8 10

Input: 60 95
Output: 60 62 64 65 66 68 70 72 74 75 76 78
        80 82 84 85 86 88 90 92 94 95
```

Approach 1 (Simple):
Hash the indices in an array from a to b and mark each of the indices as 1 or 0.
Space complexity : O(max(a, b))



Approach 2 (Better than simple):
Save memory, by translating a to 0th index and b to (b-a)th index.
Space complexity : O(|b-a|).



Simply hash |b – a| positions of an array as 0 and 1.

C++

```
 // CPP program to mark numbers as multiple of 2 or 5
#include <bits/stdc++.h>
using namespace std;

// Driver code
int main()
{
    int a = 2, b = 10;
    int size = abs(b - a) + 1;
    int* array = new int[size];

    // Iterate through a to b, If it is a multiple
    // of 2 or 5 Mark index in array as 1
    for (int i = a; i <= b; i++)
        if (i % 2 == 0 || i % 5 == 0)
            array[i - a] = 1;

    cout << "MULTIPLES of 2 and 5:\n";
    for (int i = a; i <= b; i++)
        if (array[i - a] == 1)
            cout << i << " ";

    return 0;
```

```
}
```

Java

```
 // Java program to mark numbers as
// multiple of 2 or 5
import java.lang.*;

class GFG {

    // Driver code
    public static void main(String[] args)
    {
        int a = 2, b = 10;
        int size = Math.abs(b - a) + 1;
        int array[] = new int[size];

        // Iterate through a to b, If
        // it is a multiple of 2 or 5
        // Mark index in array as 1
        for (int i = a; i <= b; i++)
            if (i % 2 == 0 || i % 5 == 0)
                array[i - a] = 1;

        System.out.println("MULTIPLES of 2"
                           + " and 5:");
        for (int i = a; i <= b; i++)
            if (array[i - a] == 1)
                System.out.printf(i + " ");
    }
}

// This code is contributed by
// Smitha Dinesh Semwal
```

Python 3

```
 # Python 3 program to mark numbers
# as multiple of 2 or 5
import math

# Driver code
a = 2
b = 10
size = abs(b - a) + 1
array = [0] * size
```

```
# Iterate through a to b,
# If it is a multiple of 2
# or 5 Mark index in array as 1
for i in range(a, b + 1):
    if (i % 2 == 0 or i % 5 == 0):
            array[i - a] = 1

print("MULTIPLES of 2 and 5:")
for i in range(a, b + 1):
    if (array[i - a] == 1):
            print(i, end=" ")

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to mark numbers as
// multiple of 2 or 5
using System;

class GFG {

    // Driver code
    static public void Main ()
    {
        int a = 2, b = 10;
        int size = Math.Abs(b - a) + 1;
        int[] array = new int[size];

        // Iterate through a to b, If
        // it is a multiple of 2 or 5
        // Mark index in array as 1
        for (int i = a; i <= b; i++)
            if (i % 2 == 0 || i % 5 == 0)
                array[i - a] = 1;

        Console.WriteLine("MULTIPLES of 2" +
                          " and 5:");
        for (int i = a; i <= b; i++)
            if (array[i - a] == 1)
                Console.Write(i + " ");
    }
}

// This code is contributed by Ajit.
```

PHP

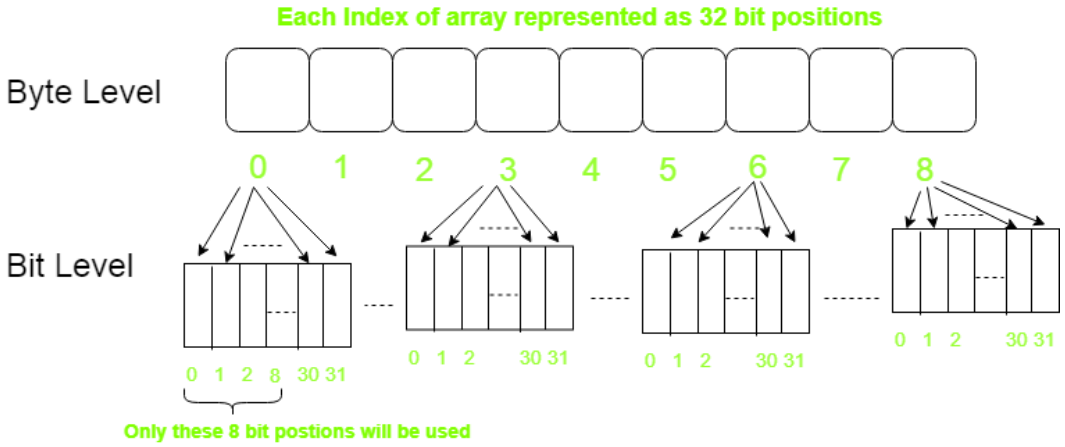
```
 <?php
// PHP program to mark
// numbers as multiple
// of 2 or 5

// Driver Code
$a = 2;
$b = 10;
$size = abs($b - $a) + 1;
$array = array_fill(0, $size, 0);

// Iterate through a to b,
// If it is a multiple of
// 2 or 5 Mark index in
// array as 1
for ($i = $a; $i <= $b; $i++)
    if ($i % 2 == 0 || $i % 5 == 0)
        $array[$i - $a] = 1;

echo "MULTIPLES of 2 and 5:\n";
for ($i = $a; $i <= $b; $i++)
    if ($array[$i - $a] == 1)
        echo $i . " ";

// This code is contributed by mits.
?>
```

Output :

```
MULTIPLES of 2 and 5:
2 4 5 6 8 10
```

Approach 3 (Using Bit Manipulations):

Here is a space optimized which uses bit manipulation technique that can be applied to problems mapping binary values in arrays.
Size of int variable in 64-bit compiler is 4 bytes. 1 byte is represented by 8 bit positions in memory. So, an integer in memory is represented by 32 bit positions(4 Bytes) these 32 bit positions can be used instead of just one index to hash binary values.



C++

```
 // CPP code to for marking multiples
#include <bits/stdc++.h>
using namespace std;

// index >> 5 corresponds to dividing index by 32
// index & 31 corresponds to modulo operation of
// index by 32

// Function to check value of bit position whether
// it is zero or one
bool checkbit(int array[], int index)
{
    return array[index >> 5] & (1 << (index & 31));
}

// Sets value of bit for corresponding index
void setbit(int array[], int index)
{
    array[index >> 5] |= (1 << (index & 31));
}

/* Driver program to test above functions*/
int main()
{
    int a = 2, b = 10;
    int size = abs(b - a);

    // Size that will be used is actual_size/32
    // ceil is used to initialize the array with
    // positive number
    size = ceil(size / 32);
```

```
    // Array is dynamically initialized as
    // we are calculating size at run time
    int* array = new int[size];

    // Iterate through every index from a to b and
    // call setbit() if it is a multiple of 2 or 5
    for (int i = a; i <= b; i++)
        if (i % 2 == 0 || i % 5 == 0)
            setbit(array, i - a);

    cout << "MULTIPLES of 2 and 5:\n";
    for (int i = a; i <= b; i++)
        if (checkbit(array, i - a))
            cout << i << " ";

    return 0;
}
```

Output:

```
MULTIPLES of 2 and 5:
2 4 5 6 8 10
```

Improved By : Smitha Dinesh Semwal, jit_t, Mithun Kumar

Source

https://www.geeksforgeeks.org/space-optimization-using-bit-manipulations/

Chapter 298

String transformation using XOR and OR

String transformation using XOR and OR - GeeksforGeeks

Given two binary strings. The task is to check if string s1 can be converted to string s2 by performing the given operations any number of times.

- Choose any two adjacent characters in a string s1 and replace one of them by a^b and the other by a | b (a OR b).

Examples:

> **Input:** S1 = “11”, S2 = “10”
> **Output:** YES
>
> Select two adjacent characters and replace s2[0] by s1[0] | s1[1] and change s2[1] by s1[0]^s1[1]
>
> **Input:** S1 = “000”, S2 = “101”
> **Output:** NO

Approach: Given below is a table which explains all the possibilities of XOR and OR operations.

X	Y	X^Y	X | Y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1

If the both the string consists of 0's only and their length is same, conversion is possible, as two adjacent zero will result in zeros only, irrespective of the operation done on it. If the both the string have 1's, follow the steps below to check if String1 can be converted to String2.

- Check if lengths are equal or not
- Check if both the strings have a minimum of one 1, as all the conversions are possible if both the strings have atleast 1 which can be seen in the table

If both of the above conditions are true, it is possible to convert String1 can be converted to String2.

Below is the implementation of above approach:

C++

```
 // C++ program to check if string1 can be
// converted to string2 using XOR and OR operations
#include <bits/stdc++.h>
using namespace std;

// function to check if conversion is possible or not
bool solve(string s1, string s2)
{
    bool flag1 = 0, flag2 = 0;

    // if lengths are different
    if (s1.length() != s2.length())
        return false;

    int l = s1.length();

    // iterate to check if both strings have 1
    for (int i = 0; i < l; i++) {

        // to check if there is
        // even one 1 in string s1
        if (s1[i] == '1')
            flag1 = 1;

        // to check if there is even
        // one 1 in string s2
        if (s2[i] == '1')
            flag2 = 1;

        if (flag1 && flag2)
            return true;
```

```
    }

    // if both string do not have a '1'.
    return false;
}

// Driver code
int main()
{
    string s1 = "100101";
    string s2 = "100000";

    if (solve(s1, s2))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // Java program to check if
// string1 can be converted
// to string2 using XOR and
// OR operations
import java.io.*;
import java.util.*;

class GFG
{

// function to check if
// conversion is possible
// or not
static boolean solve(String s1,
                     String s2)
{
    boolean flag1 = false,
            flag2 = false;

    // if lengths are different
    if (s1.length() != s2.length())
        return false;

    int l = s1.length();

    // iterate to check if
    // both strings have 1
```

```
    for (int i = 0; i < l; i++)
    {

        // to check if there is
        // even one 1 in string s1
        if (s1.charAt(i) == '1')
            flag1 = true;

        // to check if there is even
        // one 1 in string s2
        if (s2.charAt(i) == '1')
            flag2 = true;

        if (flag1 == true &&
            flag2 == true)
            return true;
    }

    // if both string do
    // not have a '1'.
    return false;
}

// Driver code
public static void main(String args[])
{
    String s1 = "100101";
    String s2 = "100000";

    if (solve(s1, s2) == true)
        System.out.print("Yes");
    else
        System.out.print("No");
}
}
```

Python3

```
 # Python3 program to check
# if string1 can be converted
# to string2 using XOR and
# OR operations

# function to check if
# conversion is possible or not
def solve(s1, s2):
    flag1 = 0
    flag2 = 0
```

```
# if lengths are different
    if (len(s1) != len(s2)):
        return False

    l = len(s1)

# iterate to check if
# both strings have 1
    for i in range (0, l):

    # to check if there is
    # even one 1 in string s1
        if (s1[i] == '1'):
            flag1 = 1;

    # to check if there is even
    # one 1 in string s2
        if (s2[i] == '1'):
            flag2 = 1

    # if both string
    # do not have a '1'.
        if (flag1 & flag2):
            return True
    return False

# Driver code
s1 = "100101"
s2 = "100000"

if solve(s1, s2):
    print( "Yes")
else:
    print("No")

# This code is contributed
# by Shivi_Aggarwal
```

C#

```
 // C# program to check if
// string1 can be converted
// to string2 using XOR and
// OR operations
using System;

class GFG
```

```
{

// function to check if
// conversion is possible
// or not
static bool solve(String s1,
                  String s2)
{
    bool flag1 = false,
         flag2 = false;

    // if lengths are different
    if (s1.Length != s2.Length)
        return false;

    int l = s1.Length;

    // iterate to check if
    // both strings have 1
    for (int i = 0; i < l; i++)
    {

        // to check if there is
        // even one 1 in string s1
        if (s1[i] == '1')
            flag1 = true;

        // to check if there is even
        // one 1 in string s2
        if (s2[i] == '1')
            flag2 = true;

        if (flag1 == true &&
            flag2 == true)
            return true;
    }

    // if both string do
    // not have a '1'.
    return false;
}

// Driver code
public static void Main()
{
    String s1 = "100101";
    String s2 = "100000";
```

```
    if (solve(s1, s2) == true)
        Console.Write("Yes");
    else
        Console.Write("No");
}
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

PHP

```
 <?php
// PHP program to check if string1
// can be converted to string2
// using XOR and OR operations

// function to check if conversion
// is possible or not
function solve($s1, $s2)
{

    // if lengths are different
    if (strlen($s1) != strlen($s2))
        return false;

    $l = strlen($s1);

    // iterate to check if
    // both strings have 1
    for ($i = 0; $i < 1; $i++)
    {

        // to check if there is
        // even one 1 in string s1
        if ($s1[$i] == '1')
            $flag1 = 1;

        // to check if there is even
        // one 1 in string s2
        if ($s2[$i] == '1')
            $flag2 = 1;

        if ($flag1 && $flag2)
            return true;
    }

    // if both string do
```

```
    // not have a '1'.
    return false;
}

// Driver code
$s1 = "100101";
$s2 = "100000";

if (solve($s1, $s2))
        echo("Yes");
    else
        echo("No");

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output:

```
Yes
```

Time Complexity: O(n) where n is length of input strings.

Improved By : Shivi_Aggarwal, Abby_akku

Source

https://www.geeksforgeeks.org/string-transformation-using-xor-and-or/

Chapter 299

Subset sum queries using bitset

Subset sum queries using bitset - GeeksforGeeks

Given an array **arr[]** and a number of queries, where in each query we have to check whether a subset whose sum is equal to given number exists in the array or not.

Examples:

```
Input : arr[]   = {1, 2, 3};
        query[] = {5, 3, 8}
Output : Yes, Yes, No
There is a subset with sum 5, subset is {2, 3}
There is a subset with sum 3, subset is {1, 2}
There is no subset with sum 8.

Input : arr[] = {4, 1, 5};
        query[] = {7, 9}
Output : No, Yes
There is no subset with sum 7.
There is a subset with sum 9, subset is {4, 5}
```

The idea is to use bitset container in C++. Using bitset, we can precalculate the existence all the subset sums in an array in O(n) and answer subsequent queries in just O(1).

We basically use an array of bits **bit[]** to represent the subset sum of elements in the array. Size of bit[] should be at least sum of all array elements plus 1 to answer all queries. We keep of bit[x] as 1 if x is a subset sum of given array, else false. Note that indexing is assumed to begin with 0.

```
For every element arr[i] of input array,
we do following
```

```
// bit[x] will be 1 if x is a subset
// sum of arr[], else 0
bit = bit | (bit << arr[i])
```

How does this work?

```
Let us consider arr[] = {3, 1, 5}, we need
to whether a subset sum of x exists or not,
where 0   x   Σarri.

We create a bitset bit[10] and reset all the
bits to 0, i.e., we make it 0000000000.

Set the 0th bit, because a subset sum of 0
exists in every array.
Now, the bit array is 0000000001

Apply the above technique for all the elements
of the array :
Current bitset = 0000000001

After doing "bit = bit | (bit << 3)",
bitset becomes    0000001001


After doing "bit | (bit << 1)",
bitset becomes    0000011011


After doing "bit | (bit << 5)",
bitset becomes    1101111011
```

Finally, we have the bit array as 1101111011, so, if bit[x] is 1 then a subset sum of x exists otherwise not. We can clearly observe that a subset sum of all the numbers from 0 to 9 except 2 and 7 exists in the array.

Here is a C++ implementation :

```
 // C++ program to answer subset sum queries using bitset
#include <bits/stdc++.h>
using namespace std;

// Maximum allowed query value
# define MAXSUM 10000

// function to check whether a subset sum equal to n
// exists in the array or not.
```

```
void processQueries(int query[], int nq, bitset<MAXSUM> bit)
{
    // One by one process subset sum queries
    for (int i=0; i<nq; i++)
    {
       int x = query[i];

       // If x is beyond size of bit[]
       if (x >= MAXSUM)
       {
           cout << "NA, ";
           continue;
       }

       // Else if x is a subset sum, then x'th bit
       // must be set
       bit[x]? cout << "Yes, " : cout << "No, ";
    }
}

// function to store all the subset sums in bit vector
void preprocess(bitset<MAXSUM> &bit, int arr[], int n)
{
    // set all the bits to 0
    bit.reset();

    // set the 0th bit because subset sum of 0 exists
    bit[0] = 1;

    // Process all array elements one by one
    for (int i = 0; i < n; ++i)

        // Do OR of following two
        // 1) All previous sums. We keep previous value
        //    of bit.
        // 2) arr[i] added to every previous sum. We
        //    move all previous indexes arr[i] ahead.
        bit |= (bit << arr[i]);
}

// Driver program
int main()
{
    int arr[] = {3, 1, 5};
    int query[] = {8, 7};

    int n  = sizeof(arr) / sizeof(arr[0]);
    int nq = sizeof(query) / sizeof(query[0]);
```

```
    // a vector of MAXSUM number of bits
    bitset<MAXSUM> bit;

    preprocess(bit, arr, n);
    processQueries(query, nq,  bit);

    return 0;
}
```

Output:

```
Yes, No,
```

Time complexity : O(n) for pre-calculating and O(1) for subsequent queries, where n is the number of elements in the array.

Refer http://stackoverflow.com/questions/12459563/what-is-the-size-of-bitset-in-c for space requirements of this approach.

Source

https://www.geeksforgeeks.org/subset-sum-queries-using-bitset/

Chapter 300

Subtract 1 without arithmetic operators

Subtract 1 without arithmetic operators - GeeksforGeeks

Write a program to subtract one from a given number. The use of operators like '+', '-', '*', '/', '++', '–' ...etc are not allowed.

Examples:

```
Input:  12
Output: 11

Input:  6
Output: 5
```

Method 1
To subtract 1 from a number x (say 0011001000), flip all the bits after the rightmost 1 bit (we get 0011001111). Finally, flip the rightmost 1 bit also (we get 0011000111) to get the answer.

C

```
 // C code to subtract
// one from a given number
#include <stdio.h>

int subtractOne(int x)
{
    int m = 1;

    // Flip all the set bits
```

```
    // until we find a 1
    while (!(x & m)) {
        x = x ^ m;
        m <<= 1;
    }

    // flip the rightmost 1 bit
    x = x ^ m;
    return x;
}

/* Driver program to test above functions*/
int main()
{
    printf("%d", subtractOne(13));
    return 0;
}
```

Java

```
 // Java code to subtract
// one from a given number
import java.io.*;

class GFG
{
static int subtractOne(int x)
{
    int m = 1;

    // Flip all the set bits
    // until we find a 1
    while (!((x & m) > 0))
    {
        x = x ^ m;
        m <<= 1;
    }

    // flip the rightmost
    // 1 bit
    x = x ^ m;
    return x;
}

// Driver Code
public static void main (String[] args)
{
    System.out.println(subtractOne(13));
```

```
}
}

// This code is contributed
// by anuj_67.
```

C#

```
 // C# code to subtract
// one from a given number
using System;

class GFG
{
static int subtractOne(int x)
{
    int m = 1;

    // Flip all the set bits
    // until we find a 1
    while (!((x & m) > 0))
    {
        x = x ^ m;
        m <<= 1;
    }

    // flip the rightmost
    // 1 bit
    x = x ^ m;
    return x;
}

// Driver Code
public static void Main ()
{
    Console.WriteLine(subtractOne(13));
}
}

// This code is contributed
// by anuj_67.
```

PHP

```
 <?php
// PHP code to subtract
// one from a given number
```

```
function subtractOne($x)
{
    $m = 1;

    // Flip all the set bits
    // until we find a 1
    while (!($x & $m))
    {
        $x = $x ^ $m;
        $m <<= 1;
    }

    // flip the
    // rightmost 1 bit
    $x = $x ^ $m;
    return $x;
}

// Driver Code
    echo subtractOne(13);

// This code is contributed
// by anuj_67.
?>
```

Output:

```
12
```

Method 2 (If + is allowed)
We know that the negative number is represented in 2's complement form on most of the architectures. We have the following lemma hold for 2's complement representation of signed numbers.

Say, x is numerical value of a number, then

~x = -(x+1) [~ is for bitwise complement]

Adding 2x on both the sides,
2x + ~x = x – 1

To obtain 2x, left shift x once.

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

```
int subtractOne(int x)
{
    return ((x << 1) + (~x));
}

/* Driver program to test above functions*/
int main()
{
    printf("%d", subtractOne(13));
    return 0;
}
```

Output:

```
12
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/subtract-1-without-arithmetic-operators/

Chapter 301

Subtract two numbers without using arithmetic operators

Subtract two numbers without using arithmetic operators - GeeksforGeeks

Write a function subtract(x, y) that returns x-y where x and y are integers. The function should not use any of the arithmetic operators (+, ++, –, -, .. etc).

The idea is to use bitwise operators. Addition of two numbers has been discussed using Bitwise operators. Like addition, the idea is to use subtractor logic.

The truth table for the half subtractor is given below.

```
X     Y     Diff     Borrow
0     0     0     0
0     1     1     1
1     0     1     0
1     1     0     0
```

From the above table one can draw the Karnaugh map for “difference” and “borrow”.

So, Logic equations are:

```
Diff   = y   x
Borrow = x' . y
```

Source: Wikipedia page for subtractor

Following is implementation based on above equations.

C

```
 // C program to Subtract two numbers
// without using arithmetic operators
#include<stdio.h>

int subtract(int x, int y)
{
    // Iterate till there
    // is no carry
    while (y != 0)
    {
        // borrow contains common
        // set bits of y and unset
        // bits of x
        int borrow = (~x) & y;

        // Subtraction of bits of x
        // and y where at least one
        // of the bits is not set
        x = x ^ y;

        // Borrow is shifted by one
        // so that subtracting it from
        // x gives the required sum
        y = borrow << 1;
    }
    return x;
}

// Driver Code
int main()
{
    int x = 29, y = 13;
    printf("x - y is %d", subtract(x, y));
    return 0;
}
```

Java

```
 // Java Program to subtract two Number
// without using arithetic operater
import java.io.*;

class GFG
{
    static int subtract(int x, int y)
    {

    // Iterate till there
```

```
    // is no carry
    while (y != 0)
    {
        // borrow contains common
        // set bits of y and unset
        // bits of x
        int borrow = (~x) & y;

        // Subtraction of bits of x
        // and y where at least one
        // of the bits is not set
        x = x ^ y;

        // Borrow is shifted by one
        // so that subtracting it from
        // x gives the required sum
        y = borrow << 1;
    }

    return x;
}

    // Driver Code
    public static void main (String[] args)
    {
        int x = 29, y = 13;

        System.out.println("x - y is " +
                        subtract(x, y));
    }
}

// This code is contributed by vt_m
```

Python3

```
 def subtract(x, y):

    # Iterate till there
    # is no carry
    while (y != 0):

        # borrow contains common
        # set bits of y and unset
        # bits of x
        borrow = (~x) & y

        # Subtraction of bits of x
```

```
        # and y where at least one
        # of the bits is not set
        x = x ^ y

        # Borrow is shifted by one
        # so that subtracting it from
        # x gives the required sum
        y = borrow << 1

    return x


# Driver Code
x = 29
y = 13
print("x - y is",subtract(x, y))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# Program to subtract two Number
// without using arithetic operater
using System;

class GFG {

    static int subtract(int x, int y)
    {

        // Iterate till there
        // is no carry
        while (y != 0)
        {

            // borrow contains common
            // set bits of y and unset
            // bits of x
            int borrow = (~x) & y;

            // Subtraction of bits of x
            // and y where at least one
            // of the bits is not set
            x = x ^ y;

            // Borrow is shifted by one
            // so that subtracting it from
```

```
            // x gives the required sum
            y = borrow << 1;
        }

        return x;
    }

    // Driver Code
    public static void Main ()
    {
        int x = 29, y = 13;

        Console.WriteLine("x - y is " +
                        subtract(x, y));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP Program to subtract two Number
// without using arithetic operater

function subtract($x, $y)
{

    // Iterate till there is no carry
    while ($y != 0)
    {

        // borrow contains common set
        // bits of y and unset
        // bits of x
        $borrow = (~$x) & $y;

        // Subtraction of bits of x
        // and y where at least
        // one of the bits is not set

        $x = $x ^ $y;

        // Borrow is shifted by one so
        // that subtracting it from
        // x gives the required sum

        $y = $borrow << 1;
```

```
    }
    return $x;
}

        // Driver Code
        $x = 29; $y = 13;
        echo "x - y is ", subtract($x,$y);

// This code is contributed by Ajit
?>
```

Output :

```
x - y is 16
```

Following is recursive implementation for the same approach.

C

```
 #include<stdio.h>

int subtract(int x, int y)
{
    if (y == 0)
        return x;
    return subtract(x ^ y, (~x & y) << 1);
}

// Driver program
int main()
{
    int x = 29, y = 13;
    printf("x - y is %d", subtract(x, y));
    return 0;
}
```

Java

```
 // Java  Program to subtract two Number
// without using arithetic operater
// Recursive implementation.
class GFG {

    static int subtract(int x, int y)
    {
```

```
        if (y == 0)
            return x;

        return subtract(x ^ y, (~x & y) << 1);
    }

    // Driver program
    public static void main(String[] args)
    {
        int x = 29, y = 13;
        System.out.printf("x - y is %d",
                          subtract(x, y));
    }
}

// This code is contributed by
// Smitha Dinesh Semwal.
```

Python3

```
 # Python  Program to
# subtract two Number
# without using arithmetic operator
# Recursive implementation.

def subtract(x, y):

    if (y == 0):
        return x
    return subtract(x ^ y, (~x & y) << 1)

# Driver program
x = 29
y = 13
print("x - y is", subtract(x, y))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# Program to subtract two Number
// without using arithetic operater
// Recursive implementation.
using System;

class GFG {
```

```
    static int subtract(int x, int y)
    {
        if (y == 0)
            return x;

        return subtract(x ^ y, (~x & y) << 1);
    }

    // Driver program
    public static void Main()
    {
        int x = 29, y = 13;
        Console.WriteLine("x - y is "+
                          subtract(x, y));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php

function subtract($x, $y)
{
    if ($y == 0)
        return $x;
    return subtract($x ^ $y,
                    (~$x & $y) << 1);
}

// Driver Code
$x = 29; $y = 13;
echo "x - y is ", subtract($x, $y);

# This code is contributed by ajit
?>
```

Output :

```
x - y is 16
```

This article is contributed **Dheeraj**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : jit_t, vt_m

Source

https://www.geeksforgeeks.org/subtract-two-numbers-without-using-arithmetic-operators/

Chapter 302

Sudo Placement | Range Queries

Sudo Placement | Range Queries - GeeksforGeeks

Given Q queries, with each query consisting of two integers L and R, the task is to find the total numbers between L and R (Both inclusive), having atmost three set bits in their binary representation.

Examples:

```
Input : Q = 2
        L = 3, R = 7
        L = 10, R = 16
Output : 5
         6
For the first query, valid numbers are 3, 4, 5, 6, and 7.
For the second query, valid numbers are 10, 11, 12, 13, 14 and 16.
```

Prerequisites : Bit Manipulation and Binary Search

Method 1 (Simple): A naive approach is to traverse all the numbers between L and R and find the number of set bits in each of those numbers. Increment a counter variable if a number does not have more than 3 set bits. Return answer as counter. **Note** : This approach is very inefficient since the numbers L and R may have large values (upto 10^{18}).

Method 2 (Efficient) : An efficient approach required here is precomputation. Since the values of L and R lie within the range [0, 10^{18}] (both inclusive), thus their binary representation can have at most 60 bits. Now, since the valid numbers are those having atmost 3 set bits, find them by generating all bit sequences of 60 bits with less than or equal to 3 set bits. This can be done by fixing, i^{th}, j^{th} and k^{th} bits for all i, j, k from (0, 60). Once, all the valid numbers are generated in sorted order, apply binary search to find the count of those numbers that lie within the given range.

Below is the implementation of above approach.

C++

```
 // CPP program to find the numbers
// having atmost 3 set bits within
// a given range
#include <bits/stdc++.h>

using namespace std;

#define LL long long int

// This function prints the required answer for each query
void answerQueries(LL Q, vector<pair<LL, LL> > query)
{
    // Set of Numbers having at most 3 set bits
    // arranged in non-descending order
    set<LL> s;

    // 0 set bits
    s.insert(0);

    // Iterate over all possible combinations of
    // i, j and k for 60 bits
    for (int i = 0; i <= 60; i++) {
        for (int j = i; j <= 60; j++) {
            for (int k = j; k <= 60; k++) {
                // 1 set bit
                if (j == i && i == k)
                    s.insert(1LL << i);

                // 2 set bits
                else if (j == k && i != j) {
                    LL x = (1LL << i) + (1LL << j);
                    s.insert(x);
                }
                else if (i == j && i != k) {
                    LL x = (1LL << i) + (1LL << k);
                    s.insert(x);
                }
                else if (i == k && i != j) {
                    LL x = (1LL << k) + (1LL << j);
                    s.insert(x);
                }

                // 3 set bits
                else {
```

```
                    LL x = (1LL << i) + (1LL << j) + (1LL << k);
                    s.insert(x);
                }
            }
        }
    }
    vector<LL> validNumbers;
    for (auto val : s)
        validNumbers.push_back(val);

    // Answer Queries by applying binary search
    for (int i = 0; i < Q; i++) {
        LL L = query[i].first;
        LL R = query[i].second;

        // Swap both the numbers if L is greater than R
        if (R < L)
            swap(L, R);
        if (L == 0)
            cout << (upper_bound(validNumbers.begin(), validNumbers.end(),
                    R) - validNumbers.begin()) << endl;
        else
            cout << (upper_bound(validNumbers.begin(), validNumbers.end(),
                   R) - upper_bound(validNumbers.begin(), validNumbers.end(),
                   L - 1)) << endl;
    }
}

// Driver Code
int main()
{
    // Number of Queries
    int Q = 2;
    vector<pair<LL, LL> > query(Q);
    query[0].first = 3;
    query[0].second = 7;
    query[1].first = 10;
    query[1].second = 16;

    answerQueries(Q, query);
    return 0;
}
```

Java

```
 // Java program to find the numbers
// having atmost 3 set bits within
// a given range
```

```
import java.util.*;
import java.io.*;

public class RangeQueries {

    //Class to store the L and R range of a query
    static class Query {
        long L;
        long R;
    }

    //It returns index of first element which is grater than searched value
     //If searched element is bigger than any array element function
     // returns first index after last element.
    public static int upperBound(ArrayList<Long> validNumbers,
                                    Long value)
    {
        int low = 0;
        int high = validNumbers.size()-1;

        while(low < high){
            int mid = (low + high)/2;
            if(value >= validNumbers.get(mid)){
                low = mid+1;
            } else {
                high = mid;
            }
        }
        return low;
    }

    public static void answerQueries(ArrayList<Query> queries){
        // Set of Numbers having at most 3 set bits
        // arranged in non-descending order
        Set<Long> allNum = new HashSet<>();

        //0 Set bits
        allNum.add(0L);

        //Iterate over all possible combinations of i, j, k for
        // 60 bits. And add all the numbers with 0, 1 or 2 set bits into
        // the set allNum.
        for(int i=0; i<=60; i++){
            for(int j=0; j<=60; j++){
                for(int k=0; k<=60; k++){

                    //For one set bit, check if i, j, k are equal
                    //if yes, then set that bit and add it to the set
```

```
            if(i==j && j==k){
                allNum.add(1L << i);
            }

            //For two set bits, two of the three variable i,j,k
            //will be equal and the third will not be. Set both
            //the bits where two variabls are equal and the bit
            //which is not equal, and add it to the set
            else if(i==j && j != k){
                long toAdd = (1L << i) + (1L << k);
                allNum.add(toAdd);
            }
            else if(i==k && k != j){
                long toAdd = (1L << i) + (1L << j);
                allNum.add(toAdd);
            }
            else if(j==k && k != i){
                long toAdd = (1L << j) + (1L << i);
                allNum.add(toAdd);
            }

            //Setting all the 3 bits
            else {
                long toAdd = (1L << i) + (1L << j) + (1L << k);
                allNum.add(toAdd);
            }

        }
    }
}

//Adding all the numbers to an array list so that it can be sorted
ArrayList<Long> validNumbers = new ArrayList<>();
for(Long num: allNum){
    validNumbers.add(num);
}

Collections.sort(validNumbers);

//Answer queries by applying binary search
for(int i=0; i<queries.size(); i++){
    long L = queries.get(i).L;
    long R = queries.get(i).R;

    //Swap L and R if R is smaller than L
    if(R < L){
        long temp = L;
        L = R;
```

```
                R = temp;
            }

            if(L == 0){
                int indxOfLastNum = upperBound(validNumbers, R);
                System.out.println(indxOfLastNum+1);
            }
            else {
                int indxOfFirstNum = upperBound(validNumbers, L);
                int indxOfLastNum = upperBound(validNumbers, R);
                System.out.println((indxOfLastNum - indxOfFirstNum +1));
            }

        }

    }

    public static void main(String[] args){
        int Q = 2;
        ArrayList<Query> queries = new ArrayList<>();

        Query q1 = new Query();
        q1.L = 3;
        q1.R = 7;

        Query q2 = new Query();
        q2.L = 10;
        q2.R = 16;

        queries.add(q1);
        queries.add(q2);

        answerQueries(queries);
    }

}
```

Time Complexity : O((Maximum Number of Bits)^3 + Q * logN), where Q is the number of queries and N is the size of set containing all valid numbers. l valid numbers.

Source

https://www.geeksforgeeks.org/sudo-placement-range-queries/

Chapter 303

Sum of Bitwise And of all pairs in a given array

Sum of Bitwise And of all pairs in a given array - GeeksforGeeks

Given an array "arr[0..n-1]" of integers, calculate sum of "arr[i] & arr[j]" for all the pairs in the given where i < j. Here & is bitwise AND operator. Expected time complexity is O(n).
Examples :

```
Input:  arr[] = {5, 10, 15}
Output: 15
Required Value = (5 & 10) + (5 & 15) + (10 & 15)
               = 0 + 5 + 10
               = 15

Input: arr[] = {1, 2, 3, 4}
Output: 3
Required Value = (1 & 2) + (1 & 3) + (1 & 4) +
                 (2 & 3) + (2 & 4) + (3 & 4)
               = 0 + 1 + 0 + 2 + 0 + 0
               = 3
```

A **Brute Force** approach is to run two loops and time complexity is $O(n^2)$.

C++

```
 // A Simple C++ program to compute sum of bitwise AND
// of all pairs
#include <bits/stdc++.h>
using namespace std;
```

```
// Returns value of "arr[0] & arr[1] + arr[0] & arr[2] +
// ... arr[i] & arr[j] + ..... arr[n-2] & arr[n-1]"
int pairAndSum(int arr[], int n)
{
    int ans = 0;  // Initialize result

    // Consider all pairs (arr[i], arr[j) such that
    // i < j
    for (int i = 0; i < n; i++)
        for (int j = i+1; j < n; j++)
           ans += arr[i] & arr[j];

    return ans;
}

// Driver program to test above function
int main()
{
    int arr[] = {5, 10, 15};
    int n = sizeof(arr) / sizeof (arr[0]);
    cout << pairAndSum(arr, n) << endl;
    return 0;
}
```

Java

```
 // A Simple Java program to compute
// sum of bitwise AND of all pairs
import java.io.*;

class GFG {

    // Returns value of "arr[0] & arr[1] +
    // arr[0] & arr[2] + ... arr[i] & arr[j] +
    // ..... arr[n-2] & arr[n-1]"
    static int pairAndSum(int arr[], int n)
    {
        int ans = 0; // Initialize result

        // Consider all pairs (arr[i], arr[j)
        // such that i < j
        for (int i = 0; i < n; i++)
            for (int j = i+1; j < n; j++)
            ans += arr[i] & arr[j];

        return ans;
    }
```

```
    // Driver program to test above function
    public static void main(String args[])
    {
        int arr[] = {5, 10, 15};
        int n = arr.length;
        System.out.println(pairAndSum(arr, n) );
    }
}

/*This code is contributed by Nikita Tiwari.*/
```

Python3

```
 # A Simple Python 3 program to compute
# sum of bitwise AND of all pairs

# Returns value of "arr[0] & arr[1] +
# arr[0] & arr[2] + ... arr[i] & arr[j] +
# ..... arr[n-2] & arr[n-1]"
def pairAndSum(arr, n) :
    ans = 0 # Initialize result

    # Consider all pairs (arr[i], arr[j)
    # such that i < j
    for i in range(0,n) :
        for j in range((i+1),n) :
            ans = ans + arr[i] & arr[j]

    return ans

# Driver program to test above function
arr = [5, 10, 15]
n = len(arr)
print(pairAndSum(arr, n))

# This code is contributed by Nikita Tiwari.
```

C#

```
 // A Simple C# program to compute
// sum of bitwise AND of all pairs
using System;

class GFG {

    // Returns value of "arr[0] & arr[1] +
```

```
    // arr[0] & arr[2] + ... arr[i] & arr[j] +
    // ..... arr[n-2] & arr[n-1]"
    static int pairAndSum(int []arr, int n)
    {

        int ans = 0; // Initialize result

        // Consider all pairs (arr[i], arr[j)
        // such that i < j
        for (int i = 0; i < n; i++)
            for (int j = i+1; j < n; j++)
                ans += arr[i] & arr[j];

        return ans;
    }

    // Driver program to test above function
    public static void Main()
    {
        int []arr = {5, 10, 15};
        int n = arr.Length;
        Console.Write(pairAndSum(arr, n) );
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// A Simple PHP program to
// compute sum of bitwise
// AND of all pairs

// Returns value of "arr[0] &
// arr[1] + arr[0] & arr[2] +
// ... arr[i] & arr[j] + .....
// arr[n-2] & arr[n-1]"

function pairAndSum($arr, $n)
{
    // Initialize result
    $ans = 0;

    // Consider all pairs (arr[i],
    // arr[j) such that i < j
    for ($i = 0; $i < $n; $i++)
        for ( $j = $i + 1; $j < $n; $j++)
```

```
        $ans += $arr[$i] & $arr[$j];

    return $ans;
}

// Driver Code
$arr = array(5, 10, 15);
$n = sizeof($arr) ;
echo pairAndSum($arr, $n), "\n";

// This code is contributed by m_kit
?>
```

Output :

```
15
```

An **Efficient Solution** can solve this problem in O(n) time. The assumption here is that integers are represented using 32 bits.

The idea is to count number of set bits at every i'th position (i>=0 && i<=31). Any i'th bit of the AND of two numbers is 1 iff the corresponding bit in both the numbers is equal to 1.

Let k be the count of set bits at i'th position. Total number of pairs with i'th set bit would be ${}^{k}C_2$ = k*(k-1)/2 (Count k means there are k numbers which have i'th set bit). Every such pair adds 2^i to total sum. Similarly, we work for all other places and add the sum to our final answer.

This idea is similar to this. Below is the implementation.

C

```
 // An efficient C++ program to compute sum of bitwise AND
// of all pairs
#include <bits/stdc++.h>
using namespace std;

// Returns value of "arr[0] & arr[1] + arr[0] & arr[2] +
// ... arr[i] & arr[j] + ..... arr[n-2] & arr[n-1]"
int pairAndSum(int arr[], int n)
{
    int ans = 0;  // Initialize result

    // Traverse over all bits
    for (int i = 0; i < 32; i++)
    {
        // Count number of elements with i'th bit set
        int k = 0;  // Initialize the count
```

```
        for (int j = 0; j < n; j++)
            if ( (arr[j] & (1 << i)) )
                k++;

        // There are k set bits, means k(k-1)/2 pairs.
        // Every pair adds 2^i to the answer. Therefore,
        // we add "2^i * [k*(k-1)/2]" to the answer.
        ans += (1<<i) * (k*(k-1)/2);
    }

    return ans;
}

// Driver program to test above function
int main()
{
    int arr[] = {5, 10, 15};
    int n = sizeof(arr) / sizeof (arr[0]);
    cout << pairAndSum(arr, n) << endl;
    return 0;
}
```

Java

```
 // An efficient Java program to compute
// sum of bitwise AND of all pairs
import java.io.*;

class GFG {

    // Returns value of "arr[0] & arr[1] +
    // arr[0] & arr[2] + ... arr[i] & arr[j] +
    // ..... arr[n-2] & arr[n-1]"
    static int pairAndSum(int arr[], int n)
    {
        int ans = 0; // Initialize result

        // Traverse over all bits
        for (int i = 0; i < 32; i++)
        {
            // Count number of elements with i'th bit set
            // Initialize the count
            int k = 0;
            for (int j = 0; j < n; j++)
            {
                if ((arr[j] & (1 << i))!=0)
                    k++;
            }
```

```
            // There are k set bits, means k(k-1)/2 pairs.
            // Every pair adds 2^i to the answer. Therefore,
            // we add "2^i * [k*(k-1)/2]" to the answer.
            ans += (1 << i) * (k * (k - 1)/2);
        }
        return ans;
    }

    // Driver program to test above function
    public static void main(String args[])
    {
        int arr[] = {5, 10, 15};
        int n = arr.length;
        System.out.println(pairAndSum(arr, n));
    }
}

/*This code is contributed by Nikita Tiwari.*/
```

Python3

```
 # An efficient Python 3 program to
# compute sum of bitwise AND of all pairs

# Returns value of "arr[0] & arr[1] +
# arr[0] & arr[2] + ... arr[i] & arr[j] +
# ..... arr[n-2] & arr[n-1]"
def pairAndSum(arr, n) :
    ans = 0 # Initialize result

    # Traverse over all bits
    for i in range(0,32) :

        # Count number of elements with i'th bit set
        # Initialize the count
        k = 0
        for j in range(0,n) :
            if ( (arr[j] & (1 << i)) ) :
                k = k + 1

        # There are k set bits, means k(k-1)/2 pairs.
        # Every pair adds 2^i to the answer. Therefore,
        # we add "2^i * [k*(k-1)/2]" to the answer.
        ans = ans + (1 << i) * (k * (k - 1) // 2)

    return ans
```

```
# Driver program to test above function
arr = [5, 10, 15]
n = len(arr)
print(pairAndSum(arr, n))

# This code is contributed by Nikita Tiwari.
```

C#

```
 // An efficient C# program to compute
// sum of bitwise AND of all pairs
using System;

class GFG {

    // Returns value of "arr[0] & arr[1] +
    // arr[0] & arr[2] + ... arr[i] & arr[j] +
    // ..... arr[n-2] & arr[n-1]"
    static int pairAndSum(int []arr, int n)
    {
        int ans = 0; // Initialize result

        // Traverse over all bits
        for (int i = 0; i < 32; i++)
        {
            // Count number of elements with
            // i'th bit set Initialize the count
            int k = 0;
            for (int j = 0; j < n; j++)
            {
                if ((arr[j] & (1 << i))!=0)
                    k++;
            }

            // There are k set bits, means
            // k(k-1)/2 pairs. Every pair
            // adds 2^i to the answer.
            // Therefore, we add "2^i *
            // [k*(k-1)/2]" to the answer.
            ans += (1 << i) * (k * (k - 1)/2);
        }

        return ans;
    }

    // Driver program to test above function
    public static void Main()
    {
```

```
        int []arr = new int[]{5, 10, 15};
        int n = arr.Length;

        Console.Write(pairAndSum(arr, n));
    }
}

/* This code is contributed by smitha*/
```

PHP

```
 <?php
// An efficient PHP program to
// compute sum of bitwise AND
// of all pairs

// Returns value of "arr[0] &
// arr[1] + arr[0] & arr[2] +
// ... arr[i] & arr[j] + .....
// arr[n-2] & arr[n-1]"
function pairAndSum($arr, $n)
{
    // Initialize result
    $ans = 0;

    // Traverse over all bits
    for ($i = 0; $i < 32; $i++)
    {

        // Count number of elements
        // with i'th bit set
        // Initialize the count
        $k = 0;
        for ($j = 0; $j < $n; $j++)
            if (($arr[$j] & (1 << $i)) )
                $k++;

        // There are k set bits,
        // means k(k-1)/2 pairs.
        // Every pair adds 2^i to
        // the answer. Therefore,
        // we add "2^i * [k*(k-1)/2]"
        // to the answer.
        $ans += (1 << $i) * ($k * ($k - 1) / 2);
    }

    return $ans;
}
```

```
    // Driver Code
    $arr = array(5, 10, 15);
    $n = sizeof($arr);
    echo pairAndSum($arr, $n) ;

// This code is contributed by nitin mittal.
?>
```

Output:

```
15
```

This article is contributed by **Ekta Goel**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : Nikita tiwari, nitin mittal, Smitha Dinesh Semwal, jit_t

Source

https://www.geeksforgeeks.org/calculate-sum-of-bitwise-and-of-all-pairs/

Chapter 304

Sum of XOR of all pairs in an array

Sum of XOR of all pairs in an array - GeeksforGeeks

Given an array of **n** integers, find the sum of xor of all pairs of numbers in the array.

Examples :

```
Input : arr[] = {7, 3, 5}
Output : 12
7 ^ 3 = 4
3 ^ 5 = 6
7 ^ 5 = 2
Sum = 4 + 6 + 2
    = 12

Input : arr[] = {5, 9, 7, 6}
Output : 47
5 ^ 9 = 12
9 ^ 7 = 14
7 ^ 6 = 1
5 ^ 7 = 2
5 ^ 6 = 3
9 ^ 6 = 15
Sum = 12 + 14 + 1 + 2 + 3 + 15
    = 47
```

Naive Solution
A **Brute Force** approach is to run two loops and time complexity is $O(n^2)$.

C++

```
 // A Simple C++ program to compute
// sum of bitwise OR of all pairs
#include <bits/stdc++.h>
using namespace std;

// Returns sum of bitwise OR
// of all pairs
int pairORSum(int arr[], int n)
{
    int ans = 0; // Initialize result

    // Consider all pairs (arr[i], arr[j) such that
    // i < j
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            ans += arr[i] ^ arr[j];

    return ans;
}

// Driver program to test above function
int main()
{
    int arr[] = { 5, 9, 7, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << pairORSum(arr, n) << endl;
    return 0;
}
```

Java

```
 // A Simple Java program to compute
// sum of bitwise OR of all pairs
import java.io.*;

class GFG {


    // Returns sum of bitwise OR
    // of all pairs
    static int pairORSum(int arr[], int n)
    {
        // Initialize result
        int ans = 0;

        // Consider all pairs (arr[i], arr[j)
        // such that i < j
        for (int i = 0; i < n; i++)
```

```
            for (int j = i + 1; j < n; j++)
                ans += arr[i] ^ arr[j];

        return ans;
    }

    // Driver program to test above function
    public static void main (String[] args) {

        int arr[] = { 5, 9, 7, 6 };
        int n = arr.length;

        System.out.println(pairORSum(arr,
                                arr.length));
    }
}


// This code is contributed by vt_m
```

Python3

```
 # A Simple Python 3 program to compute
# sum of bitwise OR of all pairs

# Returns sum of bitwise OR
# of all pairs
def pairORSum(arr, n) :
    ans = 0     # Initialize result

    # Consider all pairs (arr[i], arr[j)
    # such that i < j
    for i in range(0, n) :

        for j in range(i + 1, n) :

            ans = ans + (arr[i] ^ arr[j])

    return ans


# Driver Code
arr = [ 5, 9, 7, 6 ]
n = len(arr)

print(pairORSum(arr, n))
```

```
# This code is contributed by Nikita Tiwari.
```

C#

```
 // A Simple C# program to compute
// sum of bitwise OR of all pairs
using System;

class GFG {


    // Returns sum of bitwise OR
    // of all pairs
    static int pairORSum(int []arr, int n)
    {
        // Initialize result
        int ans = 0;

        // Consider all pairs (arr[i], arr[j)
        // such that i < j
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                ans += arr[i] ^ arr[j];

        return ans;
    }

    // Driver program to test above function
    public static void Main () {

        int []arr = { 5, 9, 7, 6 };
        int n = arr.Length;

        Console.WriteLine(pairORSum(arr,
                                arr.Length));
    }
}


// This code is contributed by vt_m
```

PHP

```
 <?php
// A Simple PHP program to compute
// sum of bitwise OR of all pairs
```

```
// Returns sum of bitwise OR
// of all pairs
function pairORSum($arr, $n)
{

    // Initialize result
    $ans = 0;

    // Consider all pairs
    // (arr[i], arr[j) such that
    // i < j
    for ( $i = 0; $i < $n; $i++)
        for ( $j = $i + 1; $j < $n; $j++)
            $ans += $arr[$i] ^ $arr[$j];

    return $ans;
}

    // Driver Code
    $arr = array( 5, 9, 7, 6 );
    $n = count($arr);
    echo pairORSum($arr, $n) ;

// This code is contributed by anuj_67.
?>
```

Output :

```
47
```

Efficient Solution
An **Efficient Solution** can solve this problem in O(n) time. The assumption here is that integers are represented using 32 bits.
Optimized solution will be to try bit manipulation. To implement the solution, we consider all bits which are 1 and which are 0 and store their count in two different variables. Next multiple those counts along with the power of 2 raised to that bit position. Do this for all the bit positions of the numbers. Their sum would be our answer.

```
Explanation :  arr[] = { 7, 3, 5 }
7 = 1 1 1
3 = 0 1 1
5 = 1 0 1
For bit position 0 :
Bits with zero = 0
Bits with one = 3
Answer = 0 * 3 * 2 ^ 0 = 0
```

```
Similarly, for bit position 1 :
Bits with zero = 1
Bits with one = 2
Answer = 1 * 2 * 2 ^ 1 = 4
Similarly, for bit position 2 :
Bits with zero = 1
Bits with one = 2
Answer = 1 * 2 * 2 ^ 2 = 8
 Final answer = 0 + 4 + 8 = 12
```

CPP

```
 // An efficient C++ program to compute
// sum of bitwise OR of all pairs
#include <bits/stdc++.h>
using namespace std;

// Returns sum of bitwise OR
// of all pairs
long long int sumXOR(int arr[], int n)
{
    long long int sum = 0;
    for (int i = 0; i < 32; i++)
    {
        //  Count of zeros and ones
        int zc = 0, oc = 0;

        // Individual sum at each bit position
        long long int idsum = 0;
        for (int j = 0; j < n; j++)
        {
            if (arr[j] % 2 == 0)
                zc++;
            else
                oc++;
            arr[j] /= 2;
        }

        // calculating individual bit sum
        idsum = oc * zc * (1 << i);
```

```
        // final sum
        sum += idsum;
    }
    return sum;
}

int main()
{
    long long int sum = 0;
    int arr[] = { 5, 9, 7, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    sum = sumXOR(arr, n);
    cout << sum;
    return 0;
}
```

Java

```
 // An efficient Java program to compute
// sum of bitwise OR of all pairs
import java.io.*;

class GFG {

    // Returns sum of bitwise OR
    // of all pairs
    static long sumXOR(int arr[], int n)
    {
        long sum = 0;
        for (int i = 0; i < 32; i++)
        {
            // Count of zeros and ones
            int zc = 0, oc = 0;

            // Individual sum at each bit position
            long idsum = 0;

            for (int j = 0; j < n; j++)
            {
                if (arr[j] % 2 == 0)
                    zc++;

                else
                    oc++;
                arr[j] /= 2;
            }

            // calculating individual bit sum
```

```
            idsum = oc * zc * (1 << i);

            // final sum
            sum += idsum;
        }
        return sum;
    }

    // Driver Code
    public static void main(String args[])
    {
        long sum = 0;
        int arr[] = { 5, 9, 7, 6 };
        int n = arr.length;

        sum = sumXOR(arr, n);
        System.out.println(sum);
    }
}

// This code is contributed by Nikita Tiwari.
```

Python3

```
 # An efficient Python3 program to compute
# sum of bitwise OR of all pair

# Returns sum of bitwise OR
# of all pairs
def sumXOR( arr,  n):

    sum = 0
    for i in range(0, 32):

        #  Count of zeros and ones
        zc = 0
        oc = 0

        # Individual sum at each bit position
        idsum = 0
        for j in range(0, n):
            if (arr[j] % 2 == 0):
                zc = zc + 1

            else:
                oc = oc + 1
            arr[j] = int(arr[j] / 2)
```

```
        # calculating individual bit sum
        idsum = oc * zc * (1 << i)

        # final sum
        sum = sum + idsum;

    return sum



# driver function
sum = 0
arr = [  5, 9, 7, 6 ]
n = len(arr)
sum = sumXOR(arr, n);
print (sum)

# This code is contributed by saloni1297
```

C#

```
 // An efficient C# program to compute
// sum of bitwise OR of all pairs
using System;

class GFG {

    // Returns sum of bitwise OR
    // of all pairs
    static long sumXOR(int []arr, int n)
    {
        long sum = 0;
        for (int i = 0; i < 32; i++)
        {
            // Count of zeros and ones
            int zc = 0, oc = 0;

            // Individual sum at each bit position
            long idsum = 0;

            for (int j = 0; j < n; j++)
            {
                if (arr[j] % 2 == 0)
                    zc++;

                else
                    oc++;
```

```
                arr[j] /= 2;
            }

            // calculating individual bit sum
            idsum = oc * zc * (1 << i);

            // final sum
            sum += idsum;
        }
        return sum;
    }

    // Driver Code
    public static void Main()
    {
        long sum = 0;
        int []arr = { 5, 9, 7, 6 };
        int n = arr.Length;

        sum = sumXOR(arr, n);
        Console.WriteLine(sum);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// An efficient PHP program to compute
// sum of bitwise OR of all pairs

// Returns sum of bitwise OR
// of all pairs
function sumXOR($arr, $n)
{
    $sum = 0;
    for ($i = 0; $i < 32; $i++)
    {
        // Count of zeros and ones
        $zc = 0; $oc = 0;

        // Individual sum at each
        // bit position
        $idsum = 0;
        for ($j = 0; $j < $n; $j++)
        {
            if ($arr[$j] % 2 == 0)
```

```
                $zc++;
            else
                $oc++;

            $arr[$j] /= 2;
        }

        // calculating individual bit sum
        $idsum = $oc * $zc * (1 << $i);

        // final sum
        $sum += $idsum;
    }

    return $sum;
}

// Driver code
    $sum = 0;
    $arr = array( 5, 9, 7, 6 );
    $n = count($arr);
    $sum = sumXOR($arr, $n);
    echo $sum;

// This code is contributed by anuj_67
?>
```

Output:

```
47
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/sum-xor-pairs-array/

Chapter 305

Sum of XOR of sum of all pairs in an array

Sum of XOR of sum of all pairs in an array - GeeksforGeeks

Given an array, find the XOR of sum of all pairs in an array.

Examples:

```
Input  : arr[] = {1, 2, 3}
Output : 0
(1 + 1) ^ (1 + 2) ^ (1 + 3) ^ (2 + 1) ^ (2 + 2) ^
(2 + 3) ^ (3 + 1) ^ (3 + 2) ^ (3 + 3) = 0

Input  : arr[] = {1, 2, 3, 4}
Output : 8
```

A **naive approach** is to consider all the pairs one by one, calculate their XOR one after the other.

C++

```
 // CPP program to find XOR of pair
// sums.
#include <bits/stdc++.h>

using namespace std;

int xorPairSum(int ar[], int n)
{
    int sum = 0;
```

```
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            sum = sum ^ (ar[i] + ar[j]);
    return sum;
}

// Driver code
int main()
{
    int arr[] = { 1, 2, 3 };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << xorPairSum(arr, n);
    return 0;
}
```

Java

```
 // Java program to find XOR of pair sums.
import java.io.*;

class GFG {

// method to find XOR of pair sums
static int xorPairSum(int ar[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            sum = sum ^ (ar[i] + ar[j]);
    return sum;
}

    // Driver code
    public static void main (String[] args)
    {
        int arr[] = {1, 2, 3};
        int n = arr.length;
        System.out.print( xorPairSum(arr, n));
    }
}

// This code is contributed by chandan_jnu.
```

Python3

```
 # Python program to find
# XOR of pair sums.
```

```
def xor_pair_sum(ar, n):
    total = 0
    for i in range(n):
        for j in range(n):
            total = total ^ (ar[i] + ar[j])

    return total


# Driver program to test the above function
if __name__ == "__main__":
    data = [1, 2, 3]
    print(xor_pair_sum(data, len(data)))

# This code is contributed
# by Kanav Malhotra
```

C#

```
 // C# program to find
// XOR of pair sums.
using System;

class GFG
{
static int xorPairSum(int []ar,
                      int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            sum = sum ^ (ar[i] + ar[j]);
    return sum;
}

// Driver code
static public void Main(String []args)
{
    int []arr = { 1, 2, 3 };
    int n = arr.Length;
    Console.WriteLine(xorPairSum(arr, n));
}
}

// This code is contributed
// by Arnab Kundu
```

PHP

```
 <?php
// PHP program to find
// XOR of pair sums.

function xorPairSum($ar, $n)
{
    $sum = 0;
    for ($i = 0; $i < $n; $i++)
        $sum = $sum ^ ($ar[$i] +
                       $ar[$j]);
    return $sum;
}

// Driver code
$arr = array( 1, 2, 3 );
$n = count($arr);
echo xorPairSum($arr, $n);

// This code is contributed
// by Subhadeep
?>
```

Output:

```
0
```

Time Complexity : $O(N^2)$

An **efficient solution** is based on XOR properties. We simply calculate the XOR of every element and then just multiply it by two.

C++

```
 // CPP program to find XOR of pair
// sums.
#include <bits/stdc++.h>

using namespace std;

int xorPairSum(int ar[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
       sum = sum ^ ar[i];
```

```
    return 2*sum;
}

// Driver code
int main()
{
    int arr[] = { 1, 2, 3 };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << xorPairSum(arr, n);
    return 0;
}
```

Java

```
 // Java program to find
// XOR of pair sums.
class GFG
{

static int xorPairSum(int ar[],
                      int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
    sum = sum ^ ar[i];
    return 2 * sum;
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 1, 2, 3 };
    int n = arr.length;
    System.out.println( xorPairSum(arr, n));
}
}

// This code is contributed
// by Arnab Kundu
```

Python3

```
 # Python3 program to find
# XOR of pair sums.

def xor_pair_sum(ar, n):
    total = 0
```

```
    for i in range(n):
        total = total ^ ar[i]

    return 2 * total


# Driver program to test the above function
if __name__ == "__main__":
    data = [1, 2, 3]
    print(xor_pair_sum(data, len(data)))

# This code is contributed
# by Kanav Malhotra
```

C#

```
 // C# program to find
// XOR of pair sums.
using System;

class GFG
{

static int xorPairSum(int []ar,
                      int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
    sum = sum ^ ar[i];
    return 2 * sum;
}

// Driver code
static public void Main(String []args)
{
    int []arr = { 1, 2, 3 };
    int n = arr.Length;
    Console.WriteLine( xorPairSum(arr, n));
}
}

// This code is contributed
// by Arnab Kundu
```

PHP

```
 <?php
```

```
// PHP program to find
// XOR of pair sums.

function xor_pair_sum($ar, $n)
{
    $total = 0;
    for($i = 0; $i < $n; $i++)
        $total = $total ^ $ar[$i];

    return (2 * $total);
}

// Driver Code
$data = array(1, 2, 3);
$n = sizeof($data);
echo xor_pair_sum($data, $n);

// This code is contributed
// by mits
?>
```

Output:

```
0
```

Time Complexity : O(N)

Improved By : Chandan_Kumar, andrew1234, kanavMalhotra, tufan_gupta2000, Mithun Kumar

Source

https://www.geeksforgeeks.org/sum-of-xor-of-sum-of-all-pairs-in-an-array/

Chapter 306

Sum of all elements up to Nth row in a Pascal triangle

Sum of all elements up to Nth row in a Pascal triangle - GeeksforGeeks

Given a row number n, and the task is to calculate the sum of all elements of each row up to n^{th} row.

Examples:

```
Input  : 2
Output : 7
Explanation:  row 0 have element 1
              row 1 have elements 1, 1
              row 2 have elements 1, 2, 1
              so, sum will be ((1) + (1 + 1) + (1 + 2 + 1)) = 7

Input  : 4
Output : 31
Explanation:  row 0 have element 1
              row 1 have elements 1, 1
              row 2 have elements 1, 2, 1
              row 3 have elements 1, 3, 3, 1
              row 4 have elements 1, 4, 6, 4, 1
              so, sum will be ((1) + (1 + 1) + (1 + 2 + 1)
              + (1 + 3 + 3 + 1) + (1 + 4 + 6 + 4 + 1)) = 31
```

Below is the example of Pascal triangle having 11 rows:

```
Pascal's triangle
```

```
0th row                              1
1st row                            1   1
2nd row                          1   2   1
3rd row                        1   3   3   1
4th row                      1   4   6   4   1
5th row                    1   5   10  10  5   1
6th row                  1   6   15  20  15  6   1
7th row                1   7   21  35  35  21  7   1
8th row              1  8   28   56  70   56  28  8  1
9th row            1   9  36  84  126  126  84  36  9  1
10th row         1  10  45  120 210  256  210 120 45 10  1
```

Naive Approach: In a Pascal triangle, each entry of a row is value of binomial coefficient. So a simple solution is to generating all row elements up to nth row and adding them. But this approach will have $O(n^3)$ time complexity. However, it can be optimized up to $O(n^2)$ time complexity. Refer the following article to generate elements of Pascal's triangle:

- Pascal's triangle

Better Solution: Let's have a look on pascal's triangle pattern

```
                                                                  sum of elements in ith row
0th row                              1                                 1    -> 20
1st row                            1   1                               2    -> 21
2nd row                          1   2   1                             4    -> 22
3rd row                        1   3   3   1                           8    -> 23
4th row                      1   4   6   4   1                         16   -> 24
5th row                    1   5   10  10  5   1                       32   -> 25
6th row                  1   6   15  20  15  6   1                     64   -> 26
7th row                1   7   21  35  35  21  7   1                   128  -> 27
8th row              1  8   28   56  70   56  28  8  1                 256  -> 28
9th row            1   9  36  84  126  126  84  36  9  1               512  -> 29
10th row         1  10  45  120 210  256  210 120 45 10  1             1024 -> 210
```

As shown above, the sum of elements in the i^{th} row is equal to 2^i. Now it can be easily calculated the sum of all elements up to n^{th} row by adding powers of 2.

Below is the implementation of above approach:

C++

```
 // C++ program to find sum of all elements
// upto nth row in Pascal triangle.
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to find sum of aal elements
// upto nth row.
long long int calculateSum(int n)
{

    // Initialize sum with 0
    long long int sum = 0;

    // Loop to calculate power of 2
    // upto n and add them
    for (int row = 0; row < n; row++) {
        sum = sum + (1 << row);
    }

    return sum;
}

// Driver function
int main()
{
    int n = 10;
    cout << " Sum of all elements:" << calculateSum(n);
    return 0;
}
```

Java

```
 // Java program to find sum of all elements
// upto nth row in Pascal triangle.
import java.io.*;

class GFG {

    // Function to find sum of aal elements
    // upto nth row.
    static long calculateSum(int n)
    {

        // Initialize sum with 0
        long sum = 0;

        // Loop to calculate power of 2
        // upto n and add them
        for (int row = 0; row < n; row++) {
            sum = sum + (1 << row);
        }
```

```
        return sum;
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 10;
        System.out.println("Sum of all elements:"
                           + calculateSum(n));
    }
}
```

Python3

```
 # Python program to find sum of all elements
# upto nth row in Pascal triangle.

# Function to find sum of aal elements
# upto nth row.
def calculateSum(n) :

    # Initialize sum with 0
    sum = 0

    # Loop to calculate power of 2
    # upto n and add them
    for row in range(n):
        sum = sum + (1 << row)

    return sum

# Driver code
n = 10
print("Sum of all elements:", calculateSum(n))
```

C#

```
 // C# program to find sum of all elements
// upto nth row in Pascal triangle.
using System;

public class GFG {

    // Function to find sum of aal elements
    // upto nth row.
    static long calculateSum(int n)
```

```
    {

        // Initialize sum with 0
        long sum = 0;

        // Loop to calculate power of 2
        // upto n and add them
        for (int row = 0; row < n; row++) {
            sum = sum + (1 << row);
        }

        return sum;
    }

    static public void Main()
    {
        int n = 10;
        Console.WriteLine("Sum of all elements:"
                          + calculateSum(n));
    }
}
```

PHP

```
 <?php
// PHP program to find sum of
// all elements upto nth row
// in Pascal triangle.

// Function to find sum of
// all elements upto nth row.
function calculateSum($n)
{

    // Initialize sum with 0
    $sum = 0;

    // Loop to calculate power
    // of 2 upto n and add them
    for ($row = 0; $row < $n; $row++)
    {
        $sum = $sum + (1 << $row);
    }

    return $sum;
}

// Driver Code
```

```
$n = 10;
echo " Sum of all elements : " .
               calculateSum($n);

// This code is contributed by Mahadev.
?>
```

Output:

```
Sum of all elements:1023
```

Time complexity: O(n)

Efficient solution:

2^n can be expressed as
$2^n = (2^0 + 2^1 + 2^2 + 2^3 + . . . + 2^{(n-1)}) + 1$

For Example:
$2^6 = (20 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5) + 1$
$64 = (1 + 2 + 4 + 8 + 16 + 32) + 1$
$64 = 63 + 1$

So, calculate 2^n instead of calculating every power of 2 up to (n – 1) and from above example the sum of the power of 2 up to (n – 1) will be $(2^n – 1)$.

C++

```
 // C++ program to find sum of all elements
// upto nth row in Pascal triangle.
#include <bits/stdc++.h>
using namespace std;

// Function to find sum of aal elements
// upto nth row.
long long int calculateSum(int n)
{

    // Initialize sum with 0
    long long int sum = 0;

    // Calculate 2^n
    sum = 1 << n;

    return (sum - 1);
}
```

```
// Driver function
int main()
{

    int n = 10;
    cout << " Sum of all elements:" << calculateSum(n);
    return 0;
}
```

Java

```
 // Java program to find sum of all elements
// upto nth row in Pascal triangle.
import java.io.*;

class GFG {

    // Function to find sum of aal elements
    // upto nth row.
    static long calculateSum(int n)
    {

        // Initialize sum with 0
        long sum = 0;

        // Calculate 2^n
        sum = 1 << n;

        return (sum - 1);
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 10;
        System.out.println("Sum of all elements:"
                           + calculateSum(n));
    }
}
```

Python3

```
 # Python3 program to find sum of all elements
# upto nth row in Pascal triangle.

# Function to find sum of aal elements
# upto nth row.
```

```
def calculateSum(n) :

    # Initialize sum with 0
    sum = 0

    # Calculate 2 ^ n
    sum = 1 << n;

    return (sum - 1)

# Driver unicode
n = 10
print("Sum of all elements:", calculateSum(n))
```

C#

```
 // C# program to find sum of all elements
// upto nth row in Pascal triangle.
using System;

public class GFG {

    // Function to find sum of aal elements
    // upto nth row.
    static long calculateSum(int n)
    {

        // Initialize sum with 0
        long sum = 0;

        // Calculate 2^n
        sum = 1 << n;

        return (sum - 1);
    }

    // Driver code
    static public void Main()
    {
        int n = 10;
        Console.WriteLine("Sum of all elements:"
                          + calculateSum(n));
    }
}
```

PHP

```
 <?php
```

```
// PHP program to find sum
// of all elements upto nth
// row in Pascal triangle.

// Function to find
// sum of all elements
// upto nth row.

function calculateSum($n)
{
    // Initialize sum with 0
    $sum = 0;

    // Calculate 2^n
    $sum = 1 << $n;

    return ($sum - 1);
}

// Driver Code
$n = 10;
echo " Sum of all elements:" ,
             calculateSum($n);

// This code is contributed
// by akt_mit
?>
```

Output:

```
Sum of all elements:1023
```

Time complexity: O(1)

Improved By : jit_t, Mahadev99

Source

https://www.geeksforgeeks.org/sum-of-all-elements-up-to-nth-row-in-a-pascals-triangle/

Chapter 307

Sum of bit differences among all pairs

Sum of bit differences among all pairs - GeeksforGeeks

Given an integer array of n integers, find sum of bit differences in all pairs that can be formed from array elements. Bit difference of a pair (x, y) is count of different bits at same positions in binary representations of x and y.
For example, bit difference for 2 and 7 is 2. Binary representation of 2 is 010 and 7 is 111 (first and last bits differ in two numbers).

Examples :

```
Input: arr[] = {1, 2}
Output: 4
All pairs in array are (1, 1), (1, 2)
                       (2, 1), (2, 2)
Sum of bit differences = 0 + 2 +
                         2 + 0
                     = 4

Input:  arr[] = {1, 3, 5}
Output: 8
All pairs in array are (1, 1), (1, 3), (1, 5)
                       (3, 1), (3, 3) (3, 5),
                       (5, 1), (5, 3), (5, 5)
Sum of bit differences =  0 + 1 + 1 +
                          1 + 0 + 2 +
                          1 + 2 + 0
                       = 8
```

Source: Google Interview Question

A **Simple Solution** is to run two loops to consider all pairs one by one. For every pair, count bit differences. Finally return sum of counts. Time complexity of this solution is $O(n^2)$.

An **Efficient Solution** can solve this problem in O(n) time using the fact that all numbers are represented using 32 bits (or some fixed number of bits). The idea is to count differences at individual bit positions. We traverse from 0 to 31 and count numbers with i'th bit set. Let this count be 'count'. There would be "n-count" numbers with i'th bit not set. So count of differences at i'th bit would be "count * (n-count) * 2".

Below is implementation of above idea.

C++

```
 // C++ program to compute sum of pairwise bit differences
#include <bits/stdc++.h>
using namespace std;

int sumBitDifferences(int arr[], int n)
{
    int ans = 0;  // Initialize result

    // traverse over all bits
    for (int i = 0; i < 32; i++)
    {
        // count number of elements with i'th bit set
        int count = 0;
        for (int j = 0; j < n; j++)
            if ( (arr[j] & (1 << i)) )
                count++;

        // Add "count * (n - count) * 2" to the answer
        ans += (count * (n - count) * 2);
    }

    return ans;
}

// Driver prorgram
int main()
{
    int arr[] = {1, 3, 5};
    int n = sizeof arr / sizeof arr[0];
    cout << sumBitDifferences(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java program to compute sum of pairwise
// bit differences

import java.io.*;

class GFG {

    static int sumBitDifferences(int arr[], int n)
    {

        int ans = 0; // Initialize result

        // traverse over all bits
        for (int i = 0; i < 32; i++) {

            // count number of elements
            // with i'th bit set
            int count = 0;

            for (int j = 0; j < n; j++)
                if ((arr[j] & (1 << i)) == 0)
                    count++;

            // Add "count * (n - count) * 2"
            // to the answer
            ans += (count * (n - count) * 2);
        }

        return ans;
    }

    // Driver prorgram
    public static void main(String args[])
    {

        int arr[] = { 1, 3, 5 };
        int n = arr.length;

        System.out.println(sumBitDifferences(
                                        arr, n));
    }
}

// This code is contributed by Anshika Goyal.
```

Python3

```
 # Python program to compute sum of pairwise bit differences
```

```
def sumBitDifferences(arr,n):

    ans = 0  # Initialize result

    # traverse over all bits
    for i in range(0, 32):

        # count number of elements with i'th bit set
        count = 0
        for j in range(0,n):
            if ( (arr[j] & (1 << i)) ):
                count+=1

        # Add "count * (n - count) * 2" to the answer
        ans += (count * (n - count) * 2);

    return ans

# Driver prorgram
arr = [1, 3, 5]
n = len(arr )
print(sumBitDifferences(arr, n))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to compute sum
// of pairwise bit differences
using System;

class GFG
{
    static int sumBitDifferences(int []arr,
                                 int n)
    {
        int ans = 0; // Initialize result

        // traverse over all bits
        for (int i = 0; i < 32; i++)
        {

            // count number of elements
            // with i'th bit set
            int count = 0;
            for (int j = 0; j < n; j++)
```

```
                if ((arr[j] & (1 << i)) == 0)
                    count++;

            // Add "count * (n - count) * 2"
            // to the answer
            ans += (count * (n - count) * 2);
        }

        return ans;
    }
    // Driver Code
    public static void Main()
    {

        int []arr = { 1, 3, 5 };
        int n = arr.Length;

        Console.Write(sumBitDifferences(arr, n));
    }
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP program to compute sum
// of pairwise bit differences

function sumBitDifferences($arr, $n)
{
    // Initialize result
    $ans = 0;

    // traverse over all bits
    for ($i = 0; $i < 32; $i++)
    {
        // count number of elements
        // with i'th bit set
        $count = 0;
        for ($j = 0; $j < $n; $j++)
            if (($arr[$j] & (1 << $i)))
                $count++;

        // Add "count * (n - count) * 2"
        // to the answer
        $ans += ($count * ($n -
                           $count) * 2);
```

```
    }

    return $ans;
}

// Driver Code
$arr = array(1, 3, 5);
$n = sizeof($arr);
echo sumBitDifferences($arr, $n), "\n";

// This code is contributed by m_kit
?>
```

Output :

```
8
```

Thanks to Gaurav Ahirwar for suggesting this solution.

Improved By : jit_t

Source

https://www.geeksforgeeks.org/sum-of-bit-differences-among-all-pairs/

Chapter 308

Sum of bitwise AND of all possible subsets of given set

Sum of bitwise AND of all possible subsets of given set - GeeksforGeeks

Given an array, we need to calculate the Sum of Bit-wise AND of all possible subsets of given array.

Examples:

```
Input : 1 2 3
Output : 9
For [1, 2, 3], all possible subsets are {1},
{2}, {3}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}
Bitwise AND of these subsets are, 1 + 2 +
3 + 0 + 1 + 2 + 0 = 9.
So, the answer would be 9.

Input : 1 2 3 4
Output : 13
```

Refer this Post for Count Set Bit
Naive Approach, we can produce all subset using Power Set then calculate Bit-wise AND sum of all subset.

A **Better** approach, we are trying to calculate which array element is responsible in producing the sum into subset.
Let's start with the least significant bit. To remove the contribution from other bits, we calculate number AND bit for all numbers in the set. Any subset of this that contain a 0 will not give any contribution. All nonempty subset that only consist of 1's will give 1 in contribution. In total there will be 2^n – 1 such subset each giving 1 in contribution. Same goes for the the other bit. We get [0, 2, 2], 3 subset each giving 2. Total 3*1 + 3*2 = 9

```
Array = {1, 2, 3}
Binary representation
positions       2 1 0
        1       0 0 1
        2       0 1 0
        3       0 1 1
              [ 0 2 2 ]
Count set bit for each position
[ 0 3 3 ] subset produced by each
position 2^n -1 i.e. n is total sum
for each position [ 0, 3*2^1, 3*2^0 ]
Now calculate the sum by multiplying
the position value i.e 2^0, 2^1 ... .
 0 + 6 + 3 = 9
```

CPP

```
 // C++ program to calculate sum of Bit-wise
// and sum of all subsets of an array
#include <bits/stdc++.h>
using namespace std;

#define BITS 32

int andSum(int arr[], int n)
{
    int ans = 0;

    // assuming representation of each element is
    // in 32 bit
    for (int i = 0; i < BITS; i++) {
        int countSetBits = 0;

        // iterating array element
        for (int j = 0; j < n; j++) {

            // Counting the set bit of array in
            // ith position
            if (arr[j] & (1 << i))
                countSetBits++;
        }

        // counting subset which produce sum when
        // particular bit position is set.
        int subset = (1 << countSetBits) - 1;

        // multiplying every position subset with 2^i
```

```
        // to count the sum.
        subset = (subset * (1 << i));

        ans += subset;
    }

    return ans;
}

// Drivers code
int main()
{
    int arr[] = { 1, 2, 3};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << andSum(arr, size);

    return 0;
}
```

Java

```
 // Java program to calculate sum of Bit-wise
// and sum of all subsets of an array
class GFG {

    static final int BITS = 32;

    static int andSum(int arr[], int n)
    {
        int ans = 0;

        // assuming representation of each
        // element is in 32 bit
        for (int i = 0; i < BITS; i++) {
            int countSetBits = 0;

            // iterating array element
            for (int j = 0; j < n; j++) {

                // Counting the set bit of
                // array in ith position
                if ((arr[j] & (1 << i)) != 0)
                    countSetBits++;
            }

            // counting subset which produce
            // sum when particular bit
            // position is set.
```

```
            int subset = (1 << countSetBits) - 1;

            // multiplying every position
            // subset with 2^i to count the
            // sum.
            subset = (subset * (1 << i));

            ans += subset;
        }

        return ans;
    }

    // Drivers code
    public static void main(String args[])
    {
        int arr[] = { 1, 2, 3};
        int size = 3;
        System.out.println (andSum(arr, size));

    }
}

// This code is contributed by Arnab Kundu.
```

Python3

```
 # Python3 program to calculate sum of
# Bit-wise and sum of all subsets of
# an array

BITS = 32;

def andSum(arr, n):
    ans = 0

    # assuming representation
    # of each element is
    # in 32 bit
    for i in range(0, BITS):
        countSetBits = 0

        # iterating array element
        for j in range(0, n) :

            # Counting the set bit
            # of array in ith
            # position
```

```
            if (arr[j] & (1 << i)) :
                countSetBits = (countSetBits
                                        + 1)

        # counting subset which
        # produce sum when
        # particular bit position
        # is set.
        subset = ((1 << countSetBits)
                                  - 1)

        # multiplying every position
        # subset with 2^i to count
        # the sum.
        subset = (subset * (1 << i))

        ans = ans + subset

    return ans

# Driver code
arr = [1, 2, 3]
size = len(arr)
print (andSum(arr, size))

# This code is contributed by
# Manish Shaw (manishshaw1)
```

C#

```
 // C# program to calculate sum of Bit-wise
// and sum of all subsets of an array
using System;

class GFG {

static int BITS = 32;

    static int andSum(int[] arr, int n)
    {
        int ans = 0;

        // assuming representation of each
        // element is in 32 bit
        for (int i = 0; i < BITS; i++) {
            int countSetBits = 0;

            // iterating array element
```

```
            for (int j = 0; j < n; j++) {

                // Counting the set bit of
                // array in ith position
                if ((arr[j] & (1 << i)) != 0)
                    countSetBits++;
            }

            // counting subset which produce
            // sum when particular bit position
            // is set.
            int subset = (1 << countSetBits) - 1;

            // multiplying every position subset
            // with 2^i to count the sum.
            subset = (subset * (1 << i));

            ans += subset;
        }

        return ans;
    }

    // Drivers code
    static public void Main()
    {
        int []arr = { 1, 2, 3};
        int size = 3;
        Console.WriteLine (andSum(arr, size));

    }
}

// This code is contributed by Arnab Kundu.
```

PHP

```
 <?php
// PHP program to calculate sum of Bit-wise
// and sum of all subsets of an array

$BITS = 32;

function andSum( $arr, $n)
{
    global $BITS;
    $ans = 0;
```

```
    // assuming representation
    // of each element is
    // in 32 bit
    for($i = 0; $i < $BITS; $i++)
    {
        $countSetBits = 0;

        // iterating array element
        for ( $j = 0; $j < $n; $j++) {

            // Counting the set bit
            // of array in ith position
            if ($arr[$j] & (1 << $i))
                $countSetBits++;
        }

        // counting subset which
        // produce sum when
        // particular bit position
        // is set.
        $subset = (1 << $countSetBits) - 1;

        // multiplying every position
        // subset with 2^i to count
        // the sum.
        $subset = ($subset * (1 << $i));

        $ans += $subset;
    }

    return $ans;
}

    // Driver code
    $arr = array(1, 2, 3);
    $size = count($arr);
    echo andSum($arr, $size);

// This code is contributed by anuj_67.
?>
```

Output:

```
9
```

Improved By : andrew1234, vt_m, manishshaw1

Source

https://www.geeksforgeeks.org/sum-bitwise-possible-subsets-given-set/

Chapter 309

Sum of bitwise OR of all possible subsets of given set

Sum of bitwise OR of all possible subsets of given set - GeeksforGeeks

Given an array arr[] of size n, we need to find sum of all the values that comes from ORing all the elements of the subsets.

Prerequisites : Subset Sum of given set

Example :

```
Input :  arr[] = {1, 2, 3}
Output : 18
Total Subsets = 23 -1= 7
1 = 1
2 = 2
3 = 3
1 | 2 = 3
1 | 3 = 3
2 | 3 = 3
1 | 2 | 3 = 3
0(empty subset)
Now SUM of all these ORs = 1 + 2 + 3 + 3 +
                           3 + 3 + 3
                         = 18

Input : arr[] = {1, 2, 3}
Output : 18
```

A **Naive approach** is to take the OR all possible combination of array[] elements and

then perform the summation of all values. **Time complexity** of this approach grows exponentially so it would not be better for large value of n.

An **Efficient** approach is to find the pattern with respect to the property of OR. Now again consider the subset in binary form like:

```
    1 = 001
    2 = 010
    3 = 011
1 | 2 = 011
1 | 3 = 011
2 | 3 = 011
1|2|3 = 011
```

Insted of taking the OR of all possible elements of array, Here we will consider all possible subset with ith bit 1.
Now, consider the ith bit in all the resultant ORs, it is zero only if all the ith bit of elements in the subset is 0.
Number of subset with ith bit 1 = total possible subsets – subsets with all ith bit 0. Here, total subsets = 2^n – 1 and subsets with all ith bits 0 = 2^(count of zeros at ith bit of all the elements of array) – 1. Now, Total subset OR with ith bit 1 = (2^n-1)-(2^(count of zeros at ith bit)-1). Total value contributed by those bits with value 1 = total subset OR with ith bit 1 *(2^i).
Now, total sum = (total subset with ith bit 1) * 2^i + (total subset with i+1th bit 1) * 2^(i+1) + …… + (total subset with 32 bit 1) * 2^32.

C++

```
 // CPP code to find the OR_SUM
#include <bits/stdc++.h>
using namespace std;

#define INT_SIZE 32

// function to find the OR_SUM
int ORsum(int arr[], int n)
{
    // create an array of size 32
    // and store the sum of bits
    // with value 0 at every index.
    int zerocnt[INT_SIZE] = { 0 };

    for (int i = 0; i < INT_SIZE; i++)
        for (int j = 0; j < n; j++)
            if (!(arr[j] & 1 << i))
                zerocnt[i] += 1;
```

```
    // for each index the OR sum contributed
    // by that bit of subset will be 2^(bit index)
    // now the OR of the bits is 0 only if
    // all the ith bit of the elements in subset
    // is 0.
    int ans = 0;
    for (int i = 0; i < INT_SIZE; i++)
    {
        ans += ((pow(2, n) - 1) -
               (pow(2, zerocnt[i]) - 1)) *
                pow(2, i);
    }

    return ans;
}

// Driver code
int main()
{
    int arr[] = { 1, 2, 3 };
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << ORsum(arr, size);
    return 0;
}
```

Java

```
 // Java code to find
// the OR_SUM
import java.io.*;

class GFG {

static int INT_SIZE = 32;

    // function to find
    // the OR_SUM
    static int ORsum(int []arr, int n)
    {

        // create an array of size 32
        // and store the sum of bits
        // with value 0 at every index.
        int zerocnt[] = new int[INT_SIZE] ;

        for (int i = 0; i < INT_SIZE; i++)
            for (int j = 0; j < n; j++)
                if ((arr[j] & 1 << i) == 0)
```

```
                    zerocnt[i] += 1;

        // for each index the OR
        // sum contributed by that
        // bit of subset will be
        // 2^(bit index) now the OR
        // of the bits is 0 only if
        // all the ith bit of the
        // elements in subset is 0.
        int ans = 0;
        for (int i = 0; i < INT_SIZE; i++)
        {
            ans += ((Math.pow(2, n) - 1) -
                (Math.pow(2, zerocnt[i]) - 1)) *
                                Math.pow(2, i);
        }

        return ans;

    }
    // Driver Code
    public static void main(String[] args)
    {
        int arr[] = { 1, 2, 3 };
        int size = arr.length;
        System.out.println(ORsum(arr, size));

    }
}

// This code is contributed by Sam007
```

Python3

```
 INT_SIZE = 32

# function to find the OR_SUM
def ORsum(arr, n):
    # create an array of size 32
    # and store the sum of bits
    # with value 0 at every index.
    zerocnt = [0 for i in range(INT_SIZE)]

    for i in range(INT_SIZE):
        for j in range(n):
            if not (arr[j] & (1 << i)):
                zerocnt[i] += 1
```

```
    # for each index the OR sum contributed
    # by that bit of subset will be 2^(bit index)
    # now the OR of the bits is 0 only if
    # all the ith bit of the elements in subset
    # is 0.
    ans = 0
    for i in range(INT_SIZE):
        ans += ((2 ** n - 1) - (2 ** zerocnt[i] - 1)) * 2 ** i

    return ans

# Driver code

if __name__ == "__main__":
    arr= [1, 2, 3]
    size = len(arr)
    print(ORsum(arr, size))

# This code is contributed by vaibhav29498
```

C#

```
 // C# code to find
// the OR_SUM
using System;

class GFG {

static int INT_SIZE = 32;

    // function to find
    // the OR_SUM
    static int ORsum(int []arr, int n)
    {

        // create an array of size 32
        // and store the sum of bits
        // with value 0 at every index.
        int []zerocnt = new int[INT_SIZE] ;

        for (int i = 0; i < INT_SIZE; i++)
            for (int j = 0; j < n; j++)
                if ((arr[j] & 1 << i) == 0)
                    zerocnt[i] += 1;

        // for each index the OR
        // sum contributed by that
        // bit of subset will be
```

```
        // 2^(bit index) now the OR
        // of the bits is 0 only if
        // all the ith bit of the
        // elements in subset is 0.
        int ans = 0;
        for (int i = 0; i < INT_SIZE; i++)
        {
            ans += (int)(((Math.Pow(2, n) - 1) -
                 (Math.Pow(2, zerocnt[i]) - 1)) *
                                Math.Pow(2, i));
        }

        return ans;

    }

    // Driver Code
    public static void Main()
    {
        int []arr = {1, 2, 3};
        int size = arr.Length;
        Console.Write(ORsum(arr, size));

    }
}

// This code is contributed by nitin mittal
```

Output:

```
18
```

Time complexity: O(n)
Auxiliary space: O(n)

Improved By : Sam007, nitin mittal, vaibhav29498

Source

https://www.geeksforgeeks.org/sum-of-bitwise-or-of-all-possible-subsets-of-given-set/

Chapter 310

Sum of bitwise OR of all subarrays

Sum of bitwise OR of all subarrays - GeeksforGeeks

Give an array of positive integers, find the total sum after performing the bit wise OR operation on all the sub arrays of a given array.

Examples:

```
Input : 1 2 3 4 5
Output : 71

Input : 6 5 4 3 2
Output : 84
```

Given an array of numbers

1	2	3	4	5

Sub array	Bitwise OR
{1}	1
{1,2}	3
{1,2,3}	3
{1,2,3,4}	7
{1,2,3,4,5}	7
{2}	2
{2,3}	3
{2,3,4}	7
{2,3,4,5]	7
{3}	3
{3,4}	7
{3,4,5}	7
{4}	4
{4,5}	5
{5}	5
Total	71

First initialize the two variable sum=0, sum1=0, variable sum will store the total sum and, with sum1 we will perform bitwise OR operation for each jth element, and add sum1 with sum.
1:- Traverse the from 0th position to n-1.
2:- For each ith variable we will perform bit wise OR operation on all the sub arrays to find the total sum.
Repeat step until the whole array is traverse.

C++

```
 // C++ program to find sum of
// bitwise ors of all subarrays.
#include <iostream>
using namespace std;

int totalSum(int a[], int n)
{
    int i, sum = 0, sum1 = 0, j;
```

```
    for (i = 0; i < n; i++)
    {

        sum1 = 0;

        // perform Bitwise OR operation
        // on all the subarray present
        // in array
        for (j = i; j < n; j++)
        {

            // OR operation
            sum1 = (sum1 | a[j]);

            // now add the sum after performing
            // the Bitwise OR operation
            sum = sum + sum1;
        }
    }

    return sum;
}

// Driver code
int main()
{
    int a[] = { 1, 2, 3, 4, 5 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << totalSum(a, n) << endl;
    return 0;
}

// This code is contributed
// by Shivi_Aggarwal
```

C

```
 // C program to find sum of bitwise ors
// of all subarrays.
#include <stdio.h>

int totalSum(int a[], int n)
{
    int i, sum = 0, sum1 = 0, j;

    for (i = 0; i < n; i++) {

        sum1 = 0;
```

```
        // perform Bitwise OR operation
        // on all the subarray present in array
        for (j = i; j < n; j++) {

            // OR operation
            sum1 = (sum1 | a[j]);

            // now add the sum after performing the
            // Bitwise OR operation
            sum = sum + sum1;
        }
    }

    return sum;
}

// Driver code
int main()
{
    int a[] = { 1, 2, 3, 4, 5 };
    int n = sizeof(a)/sizeof(a[0]);
    printf("%d ", totalSum(a, n));
    return 0;
}
```

Java

```
 // Java program to find sum
// of bitwise ors of all subarrays.
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{
static int totalSum(int a[], int n)
{
    int i, sum = 0, sum1 = 0, j;

    for (i = 0; i < n; i++)
    {
        sum1 = 0;

        // perform Bitwise OR operation
        // on all the subarray present
        // in array
        for (j = i; j < n; j++)
```

```
        {

            // OR operation
            sum1 = (sum1 | a[j]);

            // now add the sum after
            // performing the Bitwise
            // OR operation
            sum = sum + sum1;
        }
    }

    return sum;
}

// Driver code
public static void main(String args[])
{
    int a[] = { 1, 2, 3, 4, 5 };
    int n = a.length;
    System.out.println(totalSum(a,n));
}
}

// This code is contributed
// by Subhadeep
```

Python3

```
# Python3 program to find sum of
# bitwise ors of all subarrays.
def totalSum(a, n):
sum = 0;
for i in range(n):
sum1 = 0;

# perform Bitwise OR operation
# on all the subarray present
# in array
for j in range(i, n):

# OR operation
sum1 = (sum1 | a[j]);

# now add the sum after
# performing the
# Bitwise OR operation
sum = sum + sum1;
return sum;
```

```
# Driver code
a = [1, 2, 3, 4, 5];
n = len(a);
print(totalSum(a, n));

# This code is contributed by mits
```

C#

```
 // C# program to find sum
// of bitwise ors of all
// subarrays.
using System;

class GFG
{
static int totalSum(int[] a, int n)
{
    int sum = 0;
    for(int i = 0; i < n; i++)
    {
        int sum1 = 0;

        // perform Bitwise OR operation
        // on all the subarray present
        // in array
        for (int j = i; j < n; j++)
        {

            // OR operation
            sum1 = (sum1 | a[j]);

            // now add the sum after
            // performing the Bitwise
            // OR operation
            sum = sum + sum1;
        }
    }

    return sum;
}

// Driver code
static void Main()
{
    int[] a = { 1, 2, 3, 4, 5 };
    int n = a.Length;
    Console.WriteLine(totalSum(a,n));
}
```

```
}

// This code is contributed
// by mits
```

PHP

```
 <?php
// PHP program to find
// sum of bitwise ors
// of all subarrays.
function totalSum($a,$n)
{
$sum = 0;
for ($i = 0; $i < $n; $i++)
{
    $sum1 = 0;

    // perform Bitwise OR operation
    // on all the subarray present
    // in array
    for ($j = $i; $j < $n; $j++)
    {

        // OR operation
        $sum1 = ($sum1 | $a[$j]);

        // now add the sum after
        // performing the
        // Bitwise OR operation
        $sum = $sum + $sum1;
    }
}
return $sum;
}

// Driver code
$a = array(1, 2, 3, 4, 5);
$n = sizeof($a);
echo totalSum($a, $n);

// This code is contributed by mits
?>
```

Output:

```
71
```

Improved By : tufan_gupta2000, Mithun Kumar, Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/sum-of-bitwise-or-of-all-subarrays/

Chapter 311

Sum of numbers with exactly 2 bits set

Sum of numbers with exactly 2 bits set - GeeksforGeeks

Given a number n. Find sum of all number upto n whose 2 bits are set.

Examples:

```
Input : 10
Output : 33
3 + 5 + 6 + 9 + 10 = 33

Input : 100
Output : 762
```

Naive Approach: Find each number upto n whose 2 bits are set. If its 2 bits are set add it to the sum.

C++

```
 // CPP program to find sum of numbers
// upto n whose 2 bits are set
#include <bits/stdc++.h>
using namespace std;

// To count number of set bits
int countSetBits(int n)
{
    int count = 0;
    while (n) {
        n &= (n - 1);
```

```
        count++;
    }
    return count;
}

// To calculate sum of numbers
int findSum(int n)
{
    int sum = 0;

    // To count sum of number
    // whose 2 bit are set
    for (int i = 1; i <= n; i++)
        if (countSetBits(i) == 2)
            sum += i;

    return sum;
}

// Driver program to test above function
int main()
{
    int n = 10;
    cout << findSum(n);
    return 0;
}
```

Java

```
 // Java program to find sum of numbers
// upto n whose 2 bits are set
public class Main {

    // To count number of set bits
    static int countSetBits(int n)
    {
        int count = 0;
        while (n > 0) {
            n &= (n - 1);
            count++;
        }
        return count;
    }

    // To calculate sum of numbers
    static int findSum(int n)
    {
        int sum = 0;
```

```
        // To count sum of number
        // whose 2 bit are set
        for (int i = 1; i <= n; i++)
            if (countSetBits(i) == 2)
                sum += i;

        return sum;
    }

    // Driver program to test above function
    public static void main(String[] args)
    {
        int n = 10;

        System.out.println(findSum(n));
    }
}
```

Python3

```
 # Python program to find
# sum of numbers
# upto n whose 2 bits are set

# To count number of set bits
def countSetBits(n):

    count = 0
    while (n):
        n =n & (n - 1)
        count=count + 1

    return count

# To calculate sum of numbers
def findSum(n):

    sum = 0

    # To count sum of number
    # whose 2 bit are set
    for i in range(1,n+1):
        if (countSetBits(i) == 2):
            sum =sum + i

    return sum
```

```
# Driver code
n = 10
print(findSum(n))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to find sum of
// numbers upto n whose 2
// bits are set
using System;

class GFG
{

    // To count number
    // of set bits
    static int countSetBits(int n)
    {
        int count = 0;
        while (n > 0)
        {
            n = n & (n - 1);
            count++;
        }
        return count;
    }

    // To calculate
    // sum of numbers
    static int findSum(int n)
    {
        int sum = 0;

        // To count sum of number
        // whose 2 bit are set
        for (int i = 1; i <= n; i++)
            if (countSetBits(i) == 2)
                sum += i;

        return sum;
    }

    // Driver Code
    static public void Main ()
    {
```

```
        int n = 10;

        Console.WriteLine(findSum(n));
    }
}

// This code is contributed by aj_36
```

PHP

```
 <?php
// PHP program to find sum of numbers
// upto n whose 2 bits are set

// To count number of set bits
function countSetBits($n)
{
    $count = 0;
    while ($n)
    {
        $n &= ($n - 1);
        $count++;
    }
    return $count;
}

// To calculate sum of numbers
function findSum($n)
{
    $sum = 0;

    // To count sum of number
    // whose 2 bit are set
    for ($i = 1; $i <= $n; $i++)
        if (countSetBits($i) == 2)
            $sum += $i;

    return $sum;
}

    // Driver Code
    $n = 10;
    echo findSum($n);

// This code is contributed by anuj_67.
?>
```

Output:

33

Efficient Approach: The number whose 2 bits are set is of the form 2^x + 2^y and this number is less then n. So we have to find only numbers in the range upto n which is of form 2^i + 2^j where i > 0 and 2^i < n and 0 <= j < i.

C++

```
 // C++ program to find sum of numbers
// upto n whose 2 bits are set
#include <bits/stdc++.h>
using namespace std;

// To calculate sum of numbers
int findSum(int n)
{
    int sum = 0;

    // Find numbers whose 2 bits are set
    for (int i = 1; (1 << i) < n; i++) {
        for (int j = 0; j < i; j++) {
            int num = (1 << i) + (1 << j);

            // If number is greater then n
            // we don't include this in sum
            if (num <= n)
                sum += num;
        }
    }

    // Return sum of numbers
    return sum;
}

// Driver program to test findSum()
int main()
{
    int n = 10;
    cout << findSum(n);
    return 0;
}
```

Java

```
 // Java program to find sum of numbers
```

```
// upto n whose 2 bits are set
public class Main {

    // To calculate sum of numbers
    static int findSum(int n)
    {
        int sum = 0;

        // Find numbers whose 2 bits are set
        for (int i = 1; 1 << i < n; i++) {
            for (int j = 0; j < i; j++) {
                int num = (1 << i) + (1 << j);

                // If number is greater then n
                // we don't include this in sum
                if (num <= n)
                    sum += num;
            }
        }

        // Return sum of numbers
        return sum;
    }

    // Driver program to test findSum()
    public static void main(String[] args)
    {
        int n = 10;
        System.out.println(findSum(n));
    }
}
```

C#

```
 // C# program to find sum of numbers
// upto n whose 2 bits are set
using System;

public class main {

    // To calculate sum of numbers
    static int findSum(int n)
    {
        int sum = 0;

        // Find numbers whose 2 bits are set
        for (int i = 1; 1 << i < n; i++)
        {
```

```
            for (int j = 0; j < i; j++)
            {
                int num = (1 << i) + (1 << j);

                // If number is greater then n
                // we don't include this in sum
                if (num <= n)
                    sum += num;
            }
        }

        // Return sum of numbers
        return sum;
    }

    // Driver Code
    public static void Main(String []args)
    {
        int n = 10;
        Console.WriteLine(findSum(n));
    }
}

// This Code is contributed by vt_m.
```

PHP

```
 <?php
<?php
// PHP program to find sum of numbers
// upto n whose 2 bits are set

// To calculate sum of numbers
function findSum($n)
{
    $sum = 0;

    // Find numbers whose 2 bits are set
    for ($i = 1; (1 << $i) < $n; $i++)
    {
        for ($j = 0; $j < $i; $j++)
        {
            $num = (1 << $i) + (1 << $j);

            // If number is greater then n
            // we don't include this in sum
            if ($num <= $n)
                $sum += $num;
```

```
        }
    }

    // Return sum of numbers
    return $sum;
}

// Driver Code
$n = 10;
echo findSum($n);

// This code is contributed by Ajit
?>
```

Output :

```
33
```

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/sum-numbers-exactly-2-bits-set/

Chapter 312

Sum of the series 2^0 + 2^1 + 2^2 +.....+ 2^n

Sum of the series 2^0 + 2^1 + 2^2 +.....+ 2^n - GeeksforGeeks

Given an integer N, the task is to find the sum of series $2^0 + 2^1 + 2^2 + 2^3 + ... + 2^n$.

Examples:

> **Input:** 5
> **Output:** 63
> $2^0 + 2^1 + 2^2 + 2^3 + 2^4$
> = 1 + 2+ 4 + 8 + 16
> = 31
>
> **Input:** 10
> **Output:** 1023
> $2^0 + 2^1 + 2^2 + 2^3 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7 + 2^8 + 2^9$
> = 1 + 2+ 4 + 8 + 16 + 32 +64 + 128 + 256 + 512
> = 1023

A **aaive approach** is to calculate the sum is to add every power of 2 form 0 to n.

Below is the implementation of above approach:

C++

```
 // C++ program to find sum
#include <bits/stdc++.h>
using namespace std;

// function to calculate sum of series
int calculateSum(int n)
{
```

```
    // initialize sum as 0
    int sum = 0;

    // loop to calculate sum of series
    for (int i = 0; i < n; i++) {

        // calculate 2^i
        // and add it to sum

        sum = sum + (1 << i);
    }
    return sum;
}

// Driver code
int main()
{
    int n = 10;
    cout << "Sum of series of power of 2 is : "
         << calculateSum(n);
}
```

Java

```
 // Java program to find sum
class GFG {
    // function to calculate sum of series
    static int claculate sum(int n)
    {
        // initialize sum as 0
        int sum = 0;

        // loop to calculate sum of series
        for (int i = 0; i < n; i++) {

            // calculate 2^i
            // and add it to sum

            sum = sum + (1 << i);
        }
        return sum;
    }
    // Main function
    public static void main(String[] args)
    {

        int n = 10;
        System.out.println("Sum of the series : " + calculateSum(n));
```

```
    }
};
```

Python3

```
 # Python3 program to calculate
# sum of series of power of 2

# function to calculate sum of series
def calculateSum(n):
     sum = 0

     # loop to calculate sum of series
     for i in range (0, n):

         # calculate 2 ^ i
         sum = sum+ (1 << i)

     return sum

# Driver code
n = 10
print("Sum of series ", calculateSum(n))
```

C#

```
 // C# program to find sum
using System;

class GFG
{
    // function to calculate
    // sum of series
    static int calculateSum(int n)
    {
        // initialize sum as 0
        int sum = 0;

        // loop to calculate
        // sum of series
        for (int i = 0; i < n; i++)
        {

            // calculate 2^i
            // and add it to sum

            sum = sum + (1 << i);
```

```
        }
        return sum;
    }

    // Driver code
    public static void Main()
    {
        int n = 10;
        Console.WriteLine("Sum of the series : " +
                                  calculateSum(n));
    }
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

PHP

```
 <?php
// PHP program to find sum of the
// series 2^0 + 2^1 + 2^2 +…..+ 2^n

// function to calculate
// sum of series
function calculateSum($n)
{
    // initialize sum as 0
    $sum = 0;

    // loop to calculate
    // sum of series
    for ($i = 0; $i < $n; $i++)
    {

        // calculate 2^i
        // and add it to sum

        $sum = $sum + (1 << $i);
    }
    return $sum;
}

// Driver code
$n = 10;
echo "Sum of the series of " .
              "power 2 is : ",
             calculateSum($n);
```

```
// This code is contributed
// by Smitha
?>
```

Output:

```
Sum of series of power of 2 is : 1023
```

Time Complexity: O(n)

An **efficient approach** is to find the 2^n and subtract 1 from it since we know that 2^n can be written as:

```
2n = ( 20+21+22+23+24 +..... 2n-1) +1
```

Below is the implementation of above approach:

C++

```
 // C++ program to find sum
#include <bits/stdc++.h>
using namespace std;

int calculateSum(int n)
{
    // calculate  and return 2^(n+1) -1
    return (1 << (n + 1)) - 1;
}

int main()
{
    int n = 10;
    cout << "Sum of series of power of 2 is :"
         << calculateSum(n);
}
```

Java

```
 // Java program to calculate
// sum of series of power of 2

class GFG {

    // function to calculate sum of series
    static int claculate sum(int n)
    {
```

```
        // calculate 2^(n+1)
        int sum = (1 << (n + 1));
        return sum - 1;
    }

    // Driver code
    public static void main(String[] args)
    {

        int n = 10;
        System.out.println("Sum of the series of power 2 is : "
                           + calculateSum(n));
    }
};
```

Python3

```
 # Python3 program to calculate
# sum of series of 2's power

# function to calculate sum of series
def calculateSum(n):

      # calculate 2^(n + 1)
      sum = (1 << (n + 1))
      return sum-1

# Driver code
n = 10
print("Sum of series ", calculateSum(n))
```

C#

```
 // C# program to calculate
// sum of series of power of 2
using System;
class GFG
{

    // function to calculate
    // sum of series
    static int calculateSum(int n)
    {

        // calculate 2^(n+1)
        int sum = (1 << (n + 1));
```

```
        return sum - 1;
    }

    // Driver code
    public static void Main()
    {
        int n = 10;
        Console.Write("Sum of the series " +
                        "of power 2 is : " +
                           calculateSum(n));
    }

// This code is contributed
// by Smitha
}
```

PHP

```
 <?php
// PHP program to calculate
// sum of series of power of 2

// function to calculate
// sum of series
function calculateSum($n)
{

    // calculate 2^(n+1)
    $sum = (1 << ($n + 1));
    return $sum - 1;
}

// Driver code
$n = 10;
echo "Sum of the series of " .
              "power 2 is : ",
             calculateSum($n);

// This code is contributed
// by Smitha
```

Output:

```
Sum of series of power of 2 is :2047
```

Time Complexity: O(1)

Improved By : Smitha Dinesh Semwal, Abby_akku

Source

https://www.geeksforgeeks.org/sum-of-the-series-20-21-22-2n/

Chapter 313

Swap all odd and even bits

Swap all odd and even bits - GeeksforGeeks

Given an unsigned integer, swap all odd bits with even bits. For example, if the given number is 23 (**0**0**1**0**1**1**1**), it should be converted to 43 (0**0**1**0**1**0**1**1**). Every even position bit is swapped with adjacent bit on right side (even position bits are highlighted in binary representation of 23), and every odd position bit is swapped with adjacent on left side.

If we take a closer look at the example, we can observe that we basically need to right shift (>>) all even bits (In the above example, even bits of 23 are highlighted) by 1 so that they become odd bits (highlighted in 43), and left shift (<<) all odd bits by 1 so that they become even bits. The following solution is based on this observation. The solution assumes that input number is stored using 32 bits.

Let the input number be x
1) Get all even bits of x by doing bitwise and of x with 0xAAAAAAAA. The number 0xAAAAAAAA is a 32 bit number with all even bits set as 1 and all odd bits as 0.
2) Get all odd bits of x by doing bitwise and of x with 0x55555555. The number 0x55555555 is a 32 bit number with all odd bits set as 1 and all even bits as 0.
3) Right shift all even bits.
4) Left shift all odd bits.
5) Combine new even and odd bits and return.

C/C++

```
 // C program to swap even and
// odd bits of a given number
#include <stdio.h>

// Function to swap even
// and odd bits
unsigned int swapBits(unsigned int x)
{
    // Get all even bits of x
    unsigned int even_bits = x & 0xAAAAAAAA;
```

```
    // Get all odd bits of x
    unsigned int odd_bits  = x & 0x55555555;

    even_bits >>= 1;  // Right shift even bits
    odd_bits <<= 1;   // Left shift odd bits

    return (even_bits | odd_bits); // Combine even and odd bits
}

// Driver program to test above function
int main()
{
    unsigned int x = 23; // 00010111

    // Output is 43 (00101011)
    printf("%u ", swapBits(x));

    return 0;
}
```

Java

```
 // Java program to swap even
// and odd bits of a given number

class GFG{

    // Function to swap even
    // and odd bits
    static int swapBits(int x)
    {
        // Get all even bits of x
        int even_bits = x & 0xAAAAAAAA;

        // Get all odd bits of x
        int odd_bits = x & 0x55555555;

        // Right shift even bits
        even_bits >>= 1;

        // Left shift even bits
        odd_bits <<= 1;

        // Combine even and odd bits
        return (even_bits | odd_bits);
    }
```

```
    // Driver program to test above function
    public static void main(String[] args)
    {
        int x = 23; // 00010111

        // Output is 43 (00101011)
        System.out.println(swapBits(x));
    }
}

// This code is contributed by Smitha Dinesh Semwal
```

Python 3

```
 # Python 3 program to swap even
# and odd bits of a given number

# Function for swapping even
# and odd bits
def swapBits(x) :

    # Get all even bits of x
    even_bits = x & 0xAAAAAAAA

    # Get all odd bits of x
    odd_bits = x & 0x55555555

    # Right shift even bits
    even_bits >>= 1

    # Left shift odd bits
    odd_bits <<= 1

    # Combine even and odd bits
    return (even_bits | odd_bits)


# Driver program
# 00010111
x = 23

# Output is 43 (00101011)
print(swapBits(x))


# This code is contributed
# by Nikita Tiwari.
```

C#

```
 // C# program to swap even and odd bits
// of a given number
using System;

class GFG {

    // Function to swap even
    // and odd bits
    static long swapBits(int x)
    {
        // Get all even bits of x
        long even_bits = x & 0xAAAAAAAA;

        // Get all odd bits of x
        long odd_bits = x & 0x55555555;

        // Right shift even bits
        even_bits >>= 1;

        // Left shift even bits
        odd_bits <<= 1;

        // Combine even and odd bits
        return (even_bits | odd_bits);
    }

    // Driver program to test above function
    public static void Main()
    {

        int x = 23; // 00010111

        // Output is 43 (00101011)
        Console.Write(swapBits(x));
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to swap even and
// odd bits of a given number
```

```
// Function to swap even
// and odd bits
function swapBits( $x)
{

    // Get all even bits of x
    $even_bits = $x & 0xAAAAAAAA;

    // Get all odd bits of x
    $odd_bits = $x & 0x55555555;

    // Right shift even bits
    $even_bits >>= 1;

    // Left shift odd bits
    $odd_bits <<= 1;

    // Combine even and odd bits
    return ($even_bits | $odd_bits);
}

// Driver Code

// 00010111
$x = 23;

// Output is 43 (00101011)
echo swapBits($x);

// This code is contributed by Ajit
?>
```

Output:

```
 43
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/swap-all-odd-and-even-bits/

Chapter 314

Swap bits in a given number

Swap bits in a given number - GeeksforGeeks

Given a number x and two positions (from right side) in binary representation of x, write a function that swaps n bits at given two positions and returns the result. It is also given that the two sets of bits do not overlap.

Let p1 and p2 be the two given positions.

Example 1
Input:
x = 47 (00101111)
p1 = 1 (Start from second bit from right side)
p2 = 5 (Start from 6th bit from right side)
n = 3 (No of bits to be swapped)
Output:
227 (11100011)
The 3 bits starting from the second bit (from right side) are
swapped with 3 bits starting from 6th position (from right side)

Example 2
Input:
x = 28 (11100)
p1 = 0 (Start from first bit from right side)
p2 = 3 (Start from 4th bit from right side)
n = 2 (No of bits to be swapped)
Output:
7 (00111)
The 2 bits starting from 0th postion (from right side) are
swapped with 2 bits starting from 4th position (from right side)

Solution
We need to swap two sets of bits. XOR can be used in a similar way as it is used to swap 2 numbers. Following is the algorithm.

```
1) Move all bits of first set to rightmost side
   set1 =  (x >> p1) & ((1U << n) - 1)
Here the expression (1U << n) - 1 gives a number that
contains last n bits set and other bits as 0. We do &
with this expression so that bits other than the last
n bits become 0.
2) Move all bits of second set to rightmost side
   set2 =  (x >> p2) & ((1U << n) - 1)
3) XOR the two sets of bits
   xor = (set1 ^ set2)
4) Put the xor bits back to their original positions.
   xor = (xor << p1) | (xor << p2)
5) Finally, XOR the xor with original number so
   that the two sets are swapped.
   result = x ^ xor
```

Implementation:

C

```
 // C Program to swap bits
// in a given number
#include<stdio.h>

int swapBits(unsigned int x, unsigned int p1, unsigned int p2, unsigned int n)
{
    /* Move all bits of first set to rightmost side */
    unsigned int set1 =  (x >> p1) & ((1U << n) - 1);

    /* Move all bits of second set to rightmost side */
    unsigned int set2 =  (x >> p2) & ((1U << n) - 1);

    /* XOR the two sets */
    unsigned int xor = (set1 ^ set2);

    /* Put the xor bits back to their original positions */
    xor = (xor << p1) | (xor << p2);

    /* XOR the 'xor' with the original number so that the
       two sets are swapped */
    unsigned int result = x ^ xor;

    return result;
}

/* Driver program to test above function*/
int main()
```

```
{
    int res =  swapBits(28, 0, 3, 2);
    printf("\nResult = %d ", res);
    return 0;
}
```

Java

```
 //Java Program to swap bits
// in a given number

class GFG {

    static int swapBits(int x, int p1, int p2, int n)
    {
        // Move all bits of first set
        // to rightmost side
        int set1 = (x >> p1) & ((1 << n) - 1);

        // Move all bits of second set
        //to rightmost side
        int set2 = (x >> p2) & ((1 << n) - 1);

        // XOR the two sets
        int xor = (set1 ^ set2);

        // Put the xor bits back to
        // their original positions
        xor = (xor << p1) | (xor << p2);

        // XOR the 'xor' with the original number
        // so that the  two sets are swapped
        int result = x ^ xor;

        return result;
    }

    // Driver program
    public static void main(String[] args)
    {
        int res = swapBits(28, 0, 3, 2);
        System.out.println("Result = " + res);
    }
}

// This code is contributed by prerna saini.
```

Python3

```
 # Python program to
# swap bits in a given number

def swapBits(x,p1,p2,n):

    # Move all bits of first
    # set to rightmost side
    set1 =  (x >> p1) & ((1<< n) - 1)

    # Moce all bits of second
    # set to rightmost side
    set2 =  (x >> p2) & ((1 << n) - 1)

    # XOR the two sets
    xor = (set1 ^ set2)

    # Put the xor bits back
    # to their original positions
    xor = (xor << p1) | (xor << p2)

      # XOR the 'xor' with the
      # original number so that the
      # two sets are swapped
    result = x ^ xor

    return result

# Driver code

res =swapBits(28, 0, 3, 2)
print("Result =",res)

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# Program to swap bits
// in a given number
using System;

class GFG {

    static int swapBits(int x, int p1, int p2, int n)
    {
        // Move all bits of first
        //set to rightmost side
        int set1 = (x >> p1) & ((1 << n) - 1);
```

```
        // Move all bits of second set
        // set to rightmost side
        int set2 = (x >> p2) & ((1 << n) - 1);

        // XOR the two sets
        int xor = (set1 ^ set2);

        // Put the xor bits back to
        // their original positions
        xor = (xor << p1) | (xor << p2);

        // XOR the 'xor' with the original number
        // so that the two sets are swapped
        int result = x ^ xor;

        return result;
    }

    // Driver program
    public static void Main()
    {
        int res = swapBits(28, 0, 3, 2);
        Console.WriteLine("Result = " + res);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to swap bits
// in a given number

// function returns
// the swapped bits
function swapBits($x, $p1, $p2, $n)
{

    // Move all bits of first
    // set to rightmost side
    $set1 = ($x >> $p1) &
            ((1 << $n) - 1);

    // Move all bits of second
    // set to rightmost side
    $set2 = ($x >> $p2) &
```

```
            ((1 << $n) - 1);

    // XOR the two sets
    $xor = ($set1 ^ $set2);

    // Put the xor bits back to
    // their original positions
    $xor = ($xor << $p1) |
           ($xor << $p2);

    // XOR the 'xor' with the
    // original number so that
    // the two sets are swapped
    $result = $x ^ $xor;

    return $result;
}

    // Driver Code
    $res = swapBits(28, 0, 3, 2);
    echo "\nResult = ", $res;

// This code is contributed by anuj_67.
?>
```

Output:

```
 Result = 7
```

Following is a shorter implementation of the same logic

```
 int swapBits(unsigned int x, unsigned int p1, unsigned int p2, unsigned int n)
{
    /* xor contains xor of two sets */
    unsigned int xor = ((x >> p1) ^ (x >> p2)) & ((1U << n) - 1);

    /* To swap two sets, we need to again XOR the xor with original sets */
    return x ^ ((xor << p1) | (xor << p2));
}
```

References:
Swapping individual bits with XOR

Improved By : vt_m

Source

https://www.geeksforgeeks.org/swap-bits-in-a-given-number/

Chapter 315

Swap every two bits in bytes

Swap every two bits in bytes - GeeksforGeeks

Swap all the pair of bits in a byte. Before swapping: 11-10-11-01 After swapping: 11-01-11-10

Examples:

```
Input  : 00000010
Output : 00000001

Input  : 00000100
Output : 00001000
```

Approach:
x = ((x & 0b10101010) >> 1) | ((x & 0b01010101) <> 1 extracts the high bit position and shifts it to the low bit position.
Similarly the expression (x & 0b01010101) << 1 extracts the low bit from each pair and shifts it to the high bit position.
The two parts are then combined using bitwise-OR.

```
x= 00011010
((x & 0b10101010) >> 1) = 00001010 >> 1
                        = 00000101
((x & 0b01010101) << 1) = 00010000 <> 1) | ((x & 0b01010101) << 1) = 00100101
```

Below is the implementation of the above:
Note: This solution works for only 8 bit.

C++

```
 // C++ program to swap every two bits in a byte.
#include<bits/stdc++.h>
using namespace std;

unsigned int swapBitsInPair(unsigned int x)
{
    // Extracting the high bit shift it to lowbit
    // Extracting the low bit shift it to highbit
    return ((x & 0b10101010) >> 1) |
            ((x & 0b01010101) << 1);
}

/* Driver function to test above function */
int main()
{
    unsigned int x = 4;
    cout << swapBitsInPair(x);
    return 0;
}
```

Java

```
 // Java program to swap every
// two bits in a byte.
import java.util.*;

class GFG
{
    static int swapBitsInPair( int x)
    {
        // Extracting the high bit shift it to lowbit
        // Extracting the low bit shift it to highbit
        return ((x & 0b10101010) >> 1) |
                ((x & 0b01010101) << 1);
    }

    // Driver Function
    public static void main(String[] args)
    {
    int x = 4;
    System.out.print(swapBitsInPair(x));
    }
}

// This code is contributed by Gitanjali.
```

Python3

```
 # Python program to swap every
# two bits in a byte.

import math

def swapBitsInPair( x):

    # Extracting the high bit shift it to lowbit
    # Extracting the low bit shift it to highbit
    return ((x & 0b10101010) >> 1) or ((x & 0b01010101) << 1)

# driver Function
x = 4;
print(swapBitsInPair(x))

# This code is contributed by Gitanjali.
```

C#

```
 // C# program to swap every two bits in a byte.
using System;

public class GFG{

    static uint swapBitsInPair(uint x)
    {
        // Extracting the high bit shift it to lowbit
        // Extracting the low bit shift it to highbit
        return ((x & 010101010) >> 1) |
                ((x & 001010101) << 1);
    }

    // Driver function to test above function
    static public void Main () {

        uint x = 4;

        Console.WriteLine(swapBitsInPair(x));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to swap every
```

```
// two bits in a byte.

function swapBitsInPair($x)
{

    // Extracting the high bit
    // shift it to lowbit
    // Extracting the low bit
    // shift it to highbit
    return (($x & 0b10101010) >> 1) |
           (($x & 0b01010101) << 1);
}

    // Driver Code
    $x = 4;
    echo swapBitsInPair($x);

// This code is contributed by mits
?>
```

Output:

```
8
```

Reference:
https://stackoverflow.com/questions/4788799/swap-every-pair-of-bits-in-byte

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/swap-every-two-bits-bytes/

Chapter 316

Swap three variables without using temporary variable

Swap three variables without using temporary variable - GeeksforGeeks

Given three variables, a, b and c, swap them without temporary variable.

Example :

```
Input  : a = 10, b = 20 and c = 30
Output : a = 30, b = 10 and c = 20
```

Method 1 (Using Arithmetic Operators)
The idea is to get sum in one of the two given numbers. The numbers can then be swapped using the sum and subtraction from sum.
We have already discussed swapping two variables here. We can extend the same approaches

C++

```
 // C++ program to swap three variables
// without using temporary variable.
#include <iostream>
using namespace std;

// Assign c's value to a, a's value to b and
// b's value to c.
void swapThree(int &a, int &b, int &c)
{
    // Store sum of all in a
    a = a + b + c;  // (a = 60)

    // After this, b has value of a
```

```
    b = a - (b+c);  // (b = 60 - (20+30) =10)

    // After this, c has value of b
    c = a - (b+c);  // (c = 60 - (10 + 30) = 20)

    // After this, a has value of c
    a = a - (b+c);   //(a = 60 - (10 + 20) = 30)
}

// Driver code
int main()
{
    int a = 10, b = 20, c = 30;

    cout << "Before swapping a = " << a << ", b = "
         << b << ", c = " << c << endl;

    swapThree(a, b, c);

    cout << "After swapping a = " << a << ", b = "
         << b << ", c = " << c << endl;

    return 0;
}
```

C#

```
 // C# program to swap three variables
// without using temporary variable.
using System;

class GFG
{

    // Assign c's value to a, a's value
    // to b and b's value to c.
    static void swapThree(ref int a,
                          ref int b,
                          ref int c)
    {
    // Store sum of all in a
    // (a = 60)
        a = a + b + c;

    // After this, b has value of a
    // (b = 60 - (20 + 30) = 10)
        b = a - (b + c);
```

```
    // After this, c has value of b
    // (c = 60 - (10 + 30) = 20)
        c = a - (b + c);

    // After this, a has value of c
    // (a = 60 - (10 + 20) = 30)
        a = a - (b + c);
    }

    // Driver Code
    static void Main(String []args)
    {

        int a = 10, b = 20, c = 30;
        Console.WriteLine("Before swapping a = " +
                                a + ", b = " + b +
                                    ", c = " + c);

        // Calling Function
        swapThree(ref a, ref b,ref c);

        Console.Write("After swapping a = " +
                            a + ", b = " + b +
                                ", c = " + c);
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to swap three
// variables without using
// temporary variable.

// Assign c's value to a,
// a's value to b and
// b's value to c.
function swapThree(&$a, &$b, &$c)
{
    // Store sum of all in a
    $a = $a + $b + $c; // (a = 60)

    // After this, b has value of a
    // (b = 60 - (20+30) =10)
    $b = $a - ($b + $c);
```

```
    // After this, c has value of b
    // (c = 60 - (10 + 30) = 20)
    $c = $a - ($b + $c);

    // After this, a has value of c
    //(a = 60 - (10 + 20) = 30)
    $a = $a - ($b + $c);
}

// Driver Code
$a = 10; $b = 20; $c = 30;

echo "Before swapping a = " , $a ,
         ", b = ", $b , ", c = " ,
                          $c ,"\n";

swapThree($a, $b, $c);

echo "After swapping a = ",$a ,
                  ", b = ", $b ,
           ", c = ", $c , "\n";

// This code is contributed by ajit
?>
```

Output :

```
Before swapping a = 10, b = 20, c = 30
After swapping a = 30, b = 10, c = 20
```

Thanks to **Mazhar MIK** for suggesting this method.

Method 2 (Using Bitwise XOR)
The bitwise XOR operator can be used to swap three variables. The idea is similar to method 1. We first store XOR of all numbers in 'a'. Then we get individual numbers by doing XOR of this with other two numbers.

C++

```
 // C++ program to swap three variables
// without using temporary variable
#include <iostream>
using namespace std;

// Assign c's value to a, a's value to b and
// b's value to c.
```

```
void swapThree(int &a, int &b, int &c)
{
    // Store XOR of all in a
    a = a ^ b ^ c;

    // After this, b has value of a
    b = a ^ b ^ c;

    // After this, c has value of b
    c = a ^ b ^ c;

    // After this, a has value of c
    a = a ^ b ^ c;
}

// Driver code
int main()
{
    int a = 10, b = 20, c = 30;

    cout << "Before swapping a = " << a << ", b = "
         << b << ", c = " << c << endl;

    swapThree(a, b, c);

    cout << "After swapping a = " << a << ", b = "
         << b << ", c = " << c << endl;

    return 0;
}
```

C#

```
 // C# program to swap three variables
// without using temporary variable.
using System;

class GFG
{

    // Assign c's value to a, a's value
    // to b and b's value to c.
    static void swapThree(ref int a,
                          ref int b,
                          ref int c)
    {
    // Store XOR of all in a
        a = a ^ b ^ c;
```

```
    // After this, b has value of a
        b = a ^ b ^ c;

    // After this, c has value of b
        c = a ^ b ^ c;

    // After this, a has value of c
        a = a ^ b ^ c;
    }

    // Driver Code
    static void Main(String []args)
    {

    int a = 10, b = 20, c = 30;
        Console.WriteLine( "Before swapping a = " +
                            a +", b = " + b +
                               ",c = " + c);


        // Calling Function
        swapThree(ref a, ref b,ref c);


        Console.Write("After swapping a = " +
                       a +", b = " + b +
                       ", c = " + c);

    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
//PHP program to swap three variables
// without using temporary variable

// Assign c's value to a, a's value to b and
// b's value to c.

function swapThree(&$a, &$b, &$c)
{
    // Store XOR of all in a
    $a = $a ^ $b ^ $c;
```

```
    // After this, b has value of a
    $b = $a ^ $b ^ $c;

    // After this, c has value of b
    $c = $a ^ $b ^ $c;

    // After this, a has value of c
    $a = $a ^ $b ^ $c;
}

// Driver code

    $a = 10; $b = 20; $c = 30;

    echo  "Before swapping a = " , $a , ", b = ",
         $b , ", c = " , $c ,"\n";

    swapThree($a, $b, $c);

    echo "After swapping a = ", $a , ", b = ",
         $b , ", c = " , $c ,"\n";


#This code is contributed by ajit
?>
```

Output:

```
Before swapping a = 10, b = 20, c = 30
After swapping a = 30, b = 10, c = 20
```

The method 1 causes overflow for large values of a, b and c, while method 2 doesn't.

Improved By : Sam007, jit_t

Source

https://www.geeksforgeeks.org/swap-three-variables-without-using-temporary-variable/

Chapter 317

Swap two nibbles in a byte

Swap two nibbles in a byte - GeeksforGeeks

A nibbleis a four-bit aggregation, or half an octet. There are two nibbles in a byte.
Given a byte, swap the two nibbles in it. For example 100 is be represented as 01100100 in a byte (or 8 bits). The two nibbles are (0110) and (0100). If we swap the two nibbles, we get 01000110 which is 70 in decimal.

To swap the nibbles, we can use bitwise &, bitwise " operators. A byte can be represented using a unsigned char in C as size of char is 1 byte in a typical C compiler.
Below is the implementation of above idea.

C

```
 #include <stdio.h>

unsigned char swapNibbles(unsigned char x)
{
    return ( (x & 0x0F)<<4 | (x & 0xF0)>>4 );
}

int main()
{
    unsigned char x = 100;
    printf("%u", swapNibbles(x));
    return 0;
}
```

Java

```
 // Java program to swap two
// nibbles in a byte
```

```
class GFG {

static int swapNibbles(int x)
{
    return ((x & 0x0F) << 4 | (x & 0xF0) >> 4);
}

// Driver code
public static void main(String arg[])
{
    int x = 100;
    System.out.print(swapNibbles(x));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # python program Swap
# two nibbles in a byte

def swapNibbles(x):
    return ( (x & 0x0F)<<4 | (x & 0xF0)>>4 )

# Driver code

x = 100
print(swapNibbles(x))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to swap two
// nibbles in a byte
using System;

class GFG {

// Function for swapping
static int swapNibbles(int x)
{
    return ((x & 0x0F) << 4 |
            (x & 0xF0) >> 4);
}
```

```
// Driver code
public static void Main()
{
    int x = 100;
    Console.Write(swapNibbles(x));
}
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP program to swap two
// nibbles in a byte

// function to Swap two nibbles
// in a byte in php program
function swapNibbles($x)
{
    return ( ($x & 0x0F) << 4 |
           ($x & 0xF0) >> 4 );
}

    // Driver Code
    $x = 100;
    echo swapNibbles($x);

// This Code is Contributed by Ajit
?>
```

Output:

```
70
```

Explanation:
100 is 01100100 in binary. The operation can be split mainly in two parts
1) The expression "**x & 0x0F**" gives us last 4 bits of x. For x = 100, the result is 00000100. Using bitwise '<<' operator, we shift the last four bits to the left 4 times and make the new last four bits as 0. The result after shift is 01000000.

2) The expression "**x & 0xF0**" gives us first four bits of x. For x = 100, the result is 01100000. Using bitwise '>>' operator, we shift the digit to the right 4 times and make the first four bits as 0. The result after shift is 00000110.

At the end we use the bitwise OR '|' operation of the two expressions explained above. The OR operator places first nibble to the end and last nibble to first. For x = 100, the value of (01000000) OR (00000110) gives the result 01000110 which is equal to 70 in decimal.

This article is contributed by **Anuj Garg**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : nitin mittal, jit_t

Source

https://www.geeksforgeeks.org/swap-two-nibbles-byte/

Chapter 318

Toggle all bits after most significant bit

Toggle all bits after most significant bit - GeeksforGeeks

Given a number, toggle all bits of it after most significant bit including most significant bit.

Examples :

```
Input : 10
Output : 5
Binary representation of 10 is 1010
After toggling we get 0101

Input : 5
Output : 2
```

We can toggle a bit by doing XOR of it with 1 (Note that $1 \hat{} 0 = 1$ and $1 \hat{} 1 = 0$). The idea is to take a number **temp** with only one bit set. One by one move the only set bit of **temp** to left and do XOR of it with n until it crosses MSB (Most Significant Bit) of n.

C++

```
 // CPP program to toggle set  bits starting
// from MSB
#include<bits/stdc++.h>
using namespace std;

void toggle(int &n)
{
    // temporary variable to
```

```
    // use XOR with one of a n
    int temp = 1;

    // Run loop until the only
    // set bit in temp crosses
    // MST of n.
    while (temp <= n)
    {
        // Toggle bit of n
        // corresponding to
        // current set bit in
        // temp.
        n = n ^ temp;

        // Move set bit to next
        // higher position.
        temp = temp << 1;
    }
}

// Driver code
int main()
{
    int n = 10;
    toggle(n);
    cout << n;
    return 0;
}
```

Java

```
 // Java program to toggle set
// bits starting from MSB

class GFG {

static int toggle(int n) {

    // temporary variable to
    // use XOR with one of a n
    int temp = 1;

    // Run loop until the only
    // set bit in temp crosses
    // MST of n.
    while (temp <= n) {

    // Toggle bit of n
```

```
    // corresponding to
    // current set bit in
    // temp.
    n = n ^ temp;

    // Move set bit to next
    // higher position.
    temp = temp << 1;
    }
    return n;
}

// Driver code
public static void main(String arg[])
{
    int n = 10;
    n = toggle(n);
    System.out.print(n);
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python program to toggle
# set  bits starting
# from MSB

def toggle(n):

    # temporary variable to
    # use XOR with one of a n
    temp = 1

    #Run loop until the only
    #set bit in temp crosses
    #MST of n.
    while (temp <= n):

        # Toggle bit of n
        # corresponding to
        # current set bit in
        # temp.
        n = n ^ temp

        # Move set bit to next
        # higher position.
```

```
        temp = temp << 1

    return n

# Driver code

n = 10
n=toggle(n)
print(n)

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to toggle set
// bits starting from MSB
using System;

class GFG {

// Function to toggle bits
// starting from MSB
static int toggle(int n) {

    // temporary variable to
    // use XOR with one of a n
    int temp = 1;

    // Run loop until the only
    // set bit in temp crosses
    // MST of n.
    while (temp <= n) {

    // Toggle bit of n
    // corresponding to
    // current set bit in
    // temp.
    n = n ^ temp;

    // Move set bit to next
    // higher position.
    temp = temp << 1;
    }
    return n;
}

// Driver code
```

```
public static void Main()
{
    int n = 10;
    n = toggle(n);
    Console.Write(n);
}
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP program to toggle set
// bits starting from MSB

function toggle( &$n)
{
    // temporary variable to
    // use XOR with one of a n
    $temp = 1;

    // Run loop until the only
    // set bit in temp crosses
    // MST of n.
    while ($temp <= $n)
    {
        // Toggle bit of n
        // corresponding to
        // current set bit in
        // temp.
        $n = $n ^ $temp;

        // Move set bit to next
        // higher position.
        $temp = $temp << 1;
    }
}

// Driver code
$n = 10;
toggle($n);
echo $n;

// This code is contributed by ajit
?>
```

Output :

```
5
```

The above solution can be optimized to work in O(1) time under the assumption that numbers are stored in 32 bits.

C++

```
 // CPP program to toggle set  bits starting
// from MSB
#include<bits/stdc++.h>
using namespace std;

// Returns a number which has all set bits
// starting from MSB of n
int setAllBitsAfterMSB(int n)
{
    // This makes sure two bits
    // (From MSB and including MSB)
    // are set
    n |= n>>1;

    // This makes sure 4 bits
    // (From MSB and including MSB)
    // are set
    n |= n>>2;

    n |= n>>4;
    n |= n>>8;
    n |= n>>16;
    return n;
}

void toggle(int &n)
{
    n = n ^ setAllBitsAfterMSB(n);
}

// Driver code
int main()
{
    int n = 10;
    toggle(n);
    cout << n;
    return 0;
}
```

Java

```
 // Java program to toggle set bits
// starting from MSB

class GFG {

// Returns a number which has all
// set bits starting from MSB of n
static int setAllBitsAfterMSB(int n) {

    // This makes sure two bits
    // (From MSB and including MSB)
    // are set
    n |= n >> 1;

    // This makes sure 4 bits
    // (From MSB and including MSB)
    // are set
    n |= n >> 2;

    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;
    return n;
}
static int toggle(int n)
{
    n = n ^ setAllBitsAfterMSB(n);
    return n;
}

// Driver code
public static void main(String arg[])
{
    int n = 10;
    n = toggle(n);
    System.out.print(n);
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python program to toggle set  bits starting
# from MSB

# Returns a number which has all set bits
# starting from MSB of n
```

```
def setAllBitsAfterMSB(n):

    # This makes sure two bits
    # (From MSB and including MSB)
    # are set
    n |= n>>1

    # This makes sure 4 bits
    # (From MSB and including MSB)
    # are set
    n |= n>>2

    n |= n>>4
    n |= n>>8
    n |= n>>16
    return n

def toggle(n):

    n = n ^ setAllBitsAfterMSB(n)
    return n

#Driver code

n = 10
n=toggle(n)
print(n)
# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to toggle set bits
// starting from MSB
using System;

class GFG {

    // Returns a number which has all
    // set bits starting from MSB of n
    static int setAllBitsAfterMSB(int n)
    {

        // This makes sure two bits
        // (From MSB and including MSB)
        // are set
        n |= n >> 1;

        // This makes sure 4 bits
```

```
        // (From MSB and including MSB)
        // are set
        n |= n >> 2;

        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;
        return n;
    }

    static int toggle(int n)
    {
        n = n ^ setAllBitsAfterMSB(n);
        return n;
    }

    // Driver code
    public static void Main()
    {
        int n = 10;
        n = toggle(n);
        Console.WriteLine(n);
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to toggle set
// bits starting from MSB

// Returns a number which
// has all set bits starting
// from MSB of n
function setAllBitsAfterMSB($n)
{
    // This makes sure two bits
    // (From MSB and including MSB)
    // are set
    $n |= $n >> 1;

    // This makes sure 4 bits
    // (From MSB and including MSB)
    // are set
    $n |= $n >> 2;
```

```
    $n |= $n >> 4;
    $n |= $n >> 8;
    $n |= $n >> 16;
    return $n;
}
function toggle(&$n)
{
    $n = $n ^ setAllBitsAfterMSB($n);
}

// Driver Code
$n = 10;
toggle($n);
echo $n;

// This code is contributed by ajit
?>
```

Output :

```
5
```

Thanks to **Devanshu Agarwal** for suggesting this approach.

Improved By : nitin mittal, Sam007, jit_t

Source

https://www.geeksforgeeks.org/toggle-bits-significant-bit/

Chapter 319

Toggle all even bits of a number

Toggle all even bits of a number - GeeksforGeeks

Given a number, the task is to Toggle all even bit of a number

Examples:

```
Input : 10
Output : 0
binary representation 1 0 1 0
after toggle          0 0 0 0
```

```
Input : 20
Output : 30
binary representation 1 0 1 0 0
after toggle          1 1 1 1 0
```

1. First generate a number that contains even position bits.
2. Take XOR with the original number. Note that $1 \hat{} 1 = 0$ and $1 \hat{} 0 = 1$.

Let's understand this approach with below code.

C++

```
 // CPP code to Toggle all even
// bit of a number
#include <iostream>
using namespace std;

// Returns a number which has all even
// bits of n toggled.
```

```
int evenbittogglenumber(int n)
{
    // Genarate number form of 101010
    // ..till of same order as n
    int res = 0, count = 0;
    for (int temp = n; temp > 0; temp >>= 1) {

        // if bit is even then generate
        // number and or with res
        if (count % 2 == 1)
            res |= (1 << count);

        count++;
    }

    // return toggled number
    return n ^ res;
}

// Driver code
int main()
{
    int n = 11;
    cout << evenbittogglenumber(n);
    return 0;
}
```

Java

```
 // Java code to Toggle all
// even bit of a number
import java.io.*;

class GFG {

    // Returns a number which has
    // all even bits of n toggled.
    static int evenbittogglenumber(int n)
    {
        // Genarate number form of 101010
        // ..till of same order as n
        int res = 0, count = 0;
        for (int temp = n; temp > 0;
                                temp >>= 1)
        {
            // if bit is even then generate
            // number and or with res
            if (count % 2 == 1)
```

```
                res |= (1 << count);

            count++;
        }

        // return toggled number
        return n ^ res;
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 11;
        System.out.println(evenbittogglenumber(n));
    }
}

// This code is contributed by Nikita Tiwari.
```

Python3

```
 # Python code to Toggle all
# even bit of a number

# Returns a number which has all even
# bits of n toggled.
def evenbittogglenumber(n) :

    # Genarate number form of 101010
    # ..till of same order as n
    res = 0
    count = 0
    temp = n

    while (temp > 0) :

        # if bit is even then generate
        # number and or with res
        if (count % 2 == 1) :
            res = res | (1 << count)

        count = count + 1
        temp >>= 1


    # return toggled number
    return n ^ res
```

```
# Driver code
n = 11
print(evenbittogglenumber(n))

#This code is contributed by Nikita Tiwari.
```

C#

```
 // C# code to Toggle all
// even bit of a number
using System;

class GFG {

    // Returns a number which has
    // all even bits of n toggled.
    static int evenbittogglenumber(int n)
    {
        // Genarate number form of 101010
        // ..till of same order as n
        int res = 0, count = 0;

        for (int temp = n; temp > 0;
                                temp >>= 1)
        {
            // if bit is even then generate
            // number and or with res
            if (count % 2 == 1)
                res |= (1 << count);

            count++;
        }

        // return toggled number
        return n ^ res;
    }

    // Driver code
    public static void Main()
    {

        int n = 11;

        Console.WriteLine(evenbittogglenumber(n));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// php code to Toggle all
// even bit of a number

// Returns a number which has
// all even bits of n toggled.
function evenbittogglenumber($n)
{

    // Genarate number form of 101010
    // ..till of same order as n
    $res = 0;
    $count = 0;
    for ($temp = $n; $temp > 0; $temp >>= 1)
    {

        // if bit is even then generate
        // number and or with res
        if ($count % 2 == 1)
            $res |= (1 << $count);

        $count++;
    }

    // return toggled number
    return $n ^ $res;
}

    // Driver code
    $n = 11;
    echo evenbittogglenumber($n);

// This code is contributed by mits
?>
```

Output:

```
1
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/toggle-even-bits-number/

Chapter 320

Toggle all odd bits of a number

Toggle all odd bits of a number - GeeksforGeeks

Given n number, the task is to toggle odd bit of the number.

Examples:

```
Input : 10
Output : 15
binary representation 1 0 1 0
after toggle          1 1 1 1

Input : 20
Output : 1
binary representation 1 0 1 0 0
after toggle          0 0 0 0 1
```

1. First generate a number that contains odd position bits.
2. Take XOR with the original number. Note that $1 \hat{} 1 = 0$ and $1 \hat{} 0 = 1$.

Let's understand this approach with below code.

C++

```
 // Toggle all odd bit of a number
#include <iostream>
using namespace std;

// Returns a number which has all odd
// bits of n toggled.
int evenbittogglenumber(int n)
{
```

```
    // Genarate number form of 101010...
    // ..till of same order as n
    int res = 0, count = 0;
    for (int temp = n; temp > 0; temp >>= 1) {

        // if bit is odd, then generate
        // number and or with res
        if (count % 2 == 0)
            res |= (1 << count);

        count++;
    }

    // return toggled number
    return n ^ res;
}

// Driver code
int main()
{
    int n = 11;
    cout << evenbittogglenumber(n);
    return 0;
}
```

Java

```
 // Toggle all odd bit of a number

import java.io.*;

class GFG {
    // Returns a number which has all odd
    // bits of n toggled.
    static int evenbittogglenumber(int n)
    {
        // Genarate number form of 101010...
        // ..till of same order as n
        int res = 0, count = 0;
        for (int temp = n; temp > 0; temp >>= 1) {

            // if bit is odd, then generate
            // number and or with res
            if (count % 2 == 0)
                res |= (1 << count);

            count++;
        }
```

```
        // return toggled number
        return n ^ res;
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 11;
        System.out.println(evenbittogglenumber(n));
    }
}

/*This code is contributed by Nikita tiwari.*/
```

Python3

```
 # Python3 code for Toggle all odd bit of a number

# Returns a number which has all odd
# bits of n toggled.
def evenbittogglenumber(n) :

    # Genarate number form of 101010...
    # ..till of same order as n
    res = 0; count = 0; temp = n

    while(temp > 0 ) :

        # If bit is odd, then generate
        # number and or with res
        if (count % 2 == 0) :
            res = res | (1 << count)

        count = count + 1
        temp >>= 1


    # Return toggled number
    return n ^ res


# Driver code
if __name__ == '__main__' :

    n = 11
```

```
    print(evenbittogglenumber(n))


# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# code for Toggle all odd bit of a number
using System;

class GFG {

    // Returns a number which has all odd
    // bits of n toggled.
    static int evenbittogglenumber(int n)
    {

        // Genarate number form of 101010...
        // ..till of same order as n
        int res = 0, count = 0;

        for (int temp = n; temp > 0; temp >>= 1)
        {

            // if bit is odd, then generate
            // number and or with res
            if (count % 2 == 0)
                res |= (1 << count);

            count++;
        }

        // return toggled number
        return n ^ res;
    }

    // Driver code
    public static void Main()
    {

        int n = 11;

        Console.WriteLine(evenbittogglenumber(n));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// php implementation of Toggle
// all odd bit of a number

// Returns a number which has
// all odd bits of n toggled.
function evenbittogglenumber($n)
{

    // Genarate number form of 101010...
    // ..till of same order as n
    $res = 0;
    $count = 0;
    for ($temp = $n; $temp > 0; $temp >>= 1)
    {

        // if bit is odd, then generate
        // number and or with res
        if ($count % 2 == 0)
            $res |= (1 << $count);

        $count++;
    }

    // return toggled number
    return $n ^ $res;
}

    // Driver code
    $n = 11;
    echo evenbittogglenumber($n);

// This code is contributed by mits
?>
```

Output :

```
14
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/toggle-odd-bits-number/

Chapter 321

Toggle all the bits of a number except k-th bit.

Toggle all the bits of a number except k-th bit. - GeeksforGeeks

Given a positive (or unsigned) integer **n**, write a function to toggle all the bits except k-th bit. Here value of k starts from 0 (zero) and from right.

Examples:

```
Input : n = 4294967295, k = 0
Output : 1
The number 4294967295 in 32 bits has all bits
set.  When we toggle all bits except last bit,
we get 1.

Input  :  n = 1, k = 1
Output : 4294967292
4294967262 has all bits toggled except second
bit from right.
```

1. Toggle bit at k-th position. We do it by finding a number with only k-th bit set (using 1 << k), then doing bitwise XOR of this number n.
2. Toggle all bits of number obtained above using ~ (Bitwise negation)

```
 // C program to toggle all bits except kth bit
#include<stdio.h>

// Returns a number with all bit toggled in n
// except k-th bit
unsigned int toggleAllExceptK(unsigned int n,
                              unsigned int k)
```

```
{
   /* 1) Toggle k-th bit by doing n ^ (1 << k)
      2) Toggle all bits of the modified number */
    return ~(n ^ (1 << k));
}

// Driver code
int main()
{
    unsigned int n = 4294967295;
    unsigned int k = 0;
    printf("%u", toggleAllExceptK( n, k));
    return 0;
}
```

Output:

```
1
```

Source

https://www.geeksforgeeks.org/toggle-bits-number-except-k-th-bit/

Chapter 322

Toggle bits in the given range

Toggle bits in the given range - GeeksforGeeks

Given a non-negative number **n** and two values **l** and **r**. The problem is to toggle the bits in the range **l** to **r** in the binary representation of **n**, i.e, to toggle bits from the rightmost **lth** bit to the rightmost **rth** bit. A toggle operation flips a bit **0** to **1** and a bit **1** to **0**.

Constraint: 1 <= l <= r <= number of bits in the binary representation of **n**.

Examples:

```
Input : n = 17, l = 2, r = 3
Output : 23
(17)10 = (10001)2
(23)10 = (10111)2
The bits in the range 2 to 3 in the binary
representation of 17 are toggled.

Input : n = 50, l = 2, r = 5
Output : 44
```

Approach: Following are the steps:

1. Calculate **num** = ((1 << r) – 1) ^ ((1 << (l-1)) – 1). This will produce a number **num** having **r** number of bits and bits in the range **l** to **r** are the only set bits.
2. Now, perform **n = n ^ num**. This will toggle the bits in the range **l** to **r** in **n**.

C/C++

```
 // C++ implementation to toggle bits in
// the given range
#include <bits/stdc++.h>
```

```
using namespace std;

// function to toggle bits in the given range
unsigned int toggleBitsFromLToR(unsigned int n,
                                unsigned int l, unsigned int r)
{
    // calculating a number 'num' having 'r'
    // number of bits and bits in the range l
    // to r are the only set bits
    int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

    // toggle bits in the range l to r in 'n'
    // and return the number
    return (n ^ num);
}

// Driver program to test above
int main()
{
    unsigned int n = 50;
    unsigned int l = 2, r = 5;
    cout << toggleBitsFromLToR(n, l, r);
    return 0;
}
```

Java

```
 // Java implementation to toggle bits in
// the given range
import java.io.*;

class GFG
{
    // Function to toggle bits in the given range
    static int toggleBitsFromLToR(int n, int l, int r)
    {
        // calculating a number 'num' having 'r'
        // number of bits and bits in the range l
        // to r are the only set bits
        int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

        // toggle bits in the range l to r in 'n'
        // and return the number
        return (n ^ num);
    }

    // driver program
    public static void main (String[] args)
```

```
    {
        int n = 50;
        int l = 2, r = 5;
        System.out.println(toggleBitsFromLToR(n, l, r));
    }
}

// Contributed by Pramod Kumar
```

Python3

```
 # Python implementation
# to toggle bits in
# the given range

# function to toggle bits
# in the given range
def toggleBitsFromLToR(n,l,r):

    # calculating a number
    # 'num' having 'r'
    # number of bits and
    # bits in the range l
    # to r are the only set bits
    num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1)

    # toggle bits in the
    # range l to r in 'n'
    # and return the number
    return (n ^ num)

# Driver code

n = 50
l = 2
r = 5

print(toggleBitsFromLToR(n, l, r))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# implementation to toggle bits
// in the given range
using System;
```

```
namespace Toggle
{
    public class GFG
    {

    // Function to toggle bits in the given range
    static int toggleBitsFromLToR(int n, int l, int r)
    {
        // calculating a number 'num' having 'r'
        // number of bits and bits in the range l
        // to r are the only set bits
        int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

        // toggle bits in the range l to r in 'n'
        // and return the number
        return (n ^ num);
    }

    // Driver Code
    public static void Main ()
    {
        int n = 50;
        int l = 2, r = 5;
        Console.Write(toggleBitsFromLToR(n, l, r));
    }
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP implementation
// to toggle bits in
// the given range

// function to toggle bits
// in the given range
function toggleBitsFromLToR($n, $l, $r)
{

    // calculating a number
    // 'num' having 'r'
    // number of bits and
    // bits in the range l
    // to r are the only
```

```
    // set bits
    $num = ((1 << $r) - 1) ^
           ((1 << ($l - 1)) - 1);

    // toggle bits in the
    // range l to r in 'n'
    // and return the number
    return ($n ^ $num);
}

    // Driver Code
    $n = 50;
    $l = 2; $r = 5;
    echo toggleBitsFromLToR($n, $l, $r);

// This code is contributed by anuj_67
?>
```

Output:

```
44
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/toggle-bits-given-range/

Chapter 323

Toggle bits of a number except first and last bits

Toggle bits of a number except first and last bits - GeeksforGeeks

Given a number, the task is to toggle bits of the number except the first and the last bit.

Examples:

```
Input : 10
Output : 12
Binary representation:- 1 0 1 0
After toggling first and last : 1 1 0 0

Input : 9
Output : 15
Binary representation : 1 0 0 1
After toggling first and last : 1 1 1 1
```

Prerequisite : Find most significant set bit of a number

1) Generate a number which contains middle bit as set. We need to change all middle bits to 1 and keep corner bits as 0.

2) Answer is XOR of generated number and original number. Note that XOR of 1 with a number toggles the number.

C++

```
 // C++ Program to toggle bits
// except first and last bit
#include<iostream>
using namespace std;
```

```
// return set middle bits
int setmiddlebits(int n)
{

    // set all bit
    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // return middle set bits
    // shift by 1 and xor with 1
    return (n >> 1) ^ 1;
}

int togglemiddlebits(int n)
{

    // if number is 1 then
    // simply return
    if (n == 1)
        return 1;

    // xor with
    // middle bits
    return n ^ setmiddlebits(n);
}

// Driver Code
int main()
{

    // Given number
    int n = 9;

    // print toggle bits
    cout<<togglemiddlebits(n);

    return 0;
}
```

Java

```
 // Java program for toggle bits
// expect first and last bit
import java.io.*;
```

```
class GFG {

    // return set middle bits
    static int setmiddlebits(int n)
    {

        // set all bit
        n |= n >> 1;
        n |= n >> 2;
        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;

        // return middle set bits
        // shift by 1 and xor with 1
        return (n >> 1) ^ 1;
    }

    static int togglemiddlebits(int n)
    {
        // if number is 1 then
        // simply return
        if (n == 1)
            return 1;

        // XOR with middle bits
        return n ^ setmiddlebits(n);
    }

    // Driver Code
    public static void main (String[] args)
    {

        // Given number
        int n = 9;

        // print toggle bits
        System.out.println(togglemiddlebits(n));
    }
}

// This code is contributed by vt_m
```

Python3

```
 # Python3 program for toggle bits
# expect first and last bit
```

```
# return set middle bits
def setmiddlebits(n):

    # set all bit
    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    # return middle set bits
    # shift by 1 and xor with 1
    return (n >> 1) ^ 1

def togglemiddlebits(n):

    # if number is 1 then simply return
    if (n == 1):
        return 1

    # xor with middle bits
    return n ^ setmiddlebits(n)

# Driver code
n = 9
print(togglemiddlebits(n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program for toggle bits
// expect first and last bit
using System;

class GFG {

    // return set middle bits
    static int setmiddlebits(int n)
    {

        // set all bit
        n |= n >> 1;
        n |= n >> 2;
        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;
```

```
        // return middle set bits
        // shift by 1 and xor with 1
        return (n >> 1) ^ 1;
    }

    static int togglemiddlebits(int n)
    {

        // if number is 1 then
        // simply return
        if (n == 1)
            return 1;

        // XOR with middle bits
        return n ^ setmiddlebits(n);
    }

    // Driver Code
    public static void Main ()
    {

        // Given number
        int n = 9;

        // print toggle bits
        Console.WriteLine(togglemiddlebits(n));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// Php Program to toggle bits
// except first and last bit

// return set middle bits
function setmiddlebits($n)
{

    // set all bit
    $n |= $n >> 1;
    $n |= $n >> 2;
    $n |= $n >> 4;
    $n |= $n >> 8;
    $n |= $n >> 16;
```

```
    // return middle set bits
    // shift by 1 and xor with 1
    return ($n >> 1) ^ 1;
}

function togglemiddlebits($n)
{

    // if number is 1 then
    // simply return
    if ($n == 1)
        return 1;

    // xor with
    // middle bits
    return $n ^ setmiddlebits($n);
}

    // Driver Code
    $n = 9;

    // print toggle bits
    echo togglemiddlebits($n);

// This code is contributed by ajit
?>
```

Time Complexity:- O(1)
Output:

```
15
```

Improved By : Raghav Bansal, jit_t

Source

https://www.geeksforgeeks.org/toggle-bits-number-expect-first-last-bits/

Chapter 324

Toggle case of a string using Bitwise Operators

Toggle case of a string using Bitwise Operators - GeeksforGeeks

Given a string, write a function that returns toggle case of a string using the bitwise operators in place.

In ASCII codes, character 'A' is integer 65 = (0100 0001)2, while character 'a' is integer 97 = (0110 0001)2. Similarly, character 'D' is integer 68 = (0100 0100)2, while character 'd' is integer 100 = (0110 0100)2.

As we can see, only sixth least significant bit is different in ASCII code of 'A' and 'a'. Similar behavior can be seen in ASCII code of 'D' and 'd'. Therefore, we need to toggle this bit for toggling case.

Examples:

```
Input : "GeekSfOrgEEKs"
Output : "gEEKsFoRGeekS"

Input : "StRinG"
Output : "sTrINg"
```

TheASCII table is constructed in such way that the binary representation of lowercase letters is almost identical of binary representation of uppercase letters.

Toggling Case
The integer with 6th LSB as 1 is 32 (0010 0000). Therefore, bitwise XORing of a character with 32 will toggle the 6th LSB of character and hence, will toggle its case. If character is upper case, it will be converted to lower case and vice versa.

C

```
 // C program to get toggle case of a string
#include <stdio.h>

// tOGGLE cASE = swaps CAPS to lower
// case and lower case to CAPS
char *toggleCase(char *a)
{
    for (int i=0; a[i]!='\0'; i++) {

        // Bitwise EXOR with 32
        a[i] ^= 32;
    }

    return a;
}

// Driver Code
int main()
{
    char str[] = "CheRrY";
    printf("Toggle case: %s\n", toggleCase(str));
    printf("Original string: %s", toggleCase(str));
    return 0;
}
```

Java

```
 // program to get toggle case of a string

public class Test
{

    static int x=32;

    // tOGGLE cASE = swaps CAPS to lower
    // case and lower case to CAPS
    static String toggleCase(char[] a)
    {
        for (int i=0; i<a.length; i++) {

            // Bitwise EXOR with 32
            a[i]^=32;
        }
        return new String(a);
    }

    /* Driver program */
    public static void main(String[] args)
```

```
    {
        String str = "CheRrY";
        System.out.print("Toggle case: ");
        str = toggleCase(str.toCharArray());
        System.out.println(str);

        System.out.print("Original string: ");
        str = toggleCase(str.toCharArray());
        System.out.println(str);
    }
}
```

C#

```
 // C# program to get toggle case of a string
using System;

class GFG {

    // tOGGLE cASE = swaps CAPS to lower
    // case and lower case to CAPS
    static string toggleCase(char []a)
    {
        for (int i = 0; i < a.Length; i++)
        {

            // Bitwise EXOR with 32
            a[i] ^= (char)32;
        }

        return new string(a);
    }

    /* Driver program */
    public static void Main()
    {
        string str = "CheRrY";
        Console.Write("Toggle case: ");
        str = toggleCase(str.ToCharArray());
        Console.WriteLine(str);

        Console.Write("Original string: ");
        str = toggleCase(str.ToCharArray());
        Console.Write(str);
    }
}

// This code is contributed by nitin mittal.
```

Output:

```
Toggle case: cHErRy
Original string: CheRrY
```

Thanks to Kumar Gaurav for improving the solution.

Similar Article :
Case conversion of a string using BitWise operators in C/C++

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/toggle-case-string-using-bitwise-operators/

Chapter 325

Toggle first and last bits of a number

Toggle first and last bits of a number - GeeksforGeeks

Given a number n, the task is to toggle only first and last bits of a number

Examples :

```
Input : 10
Output : 3
Binary representation of 10 is
1010. After toggling first and
last bits, we get 0011.

Input : 15
Output : 6
```

Prerequisite : Find MSB of given number.

1) Generate a number which contains first and last bit as set. We need to change all middle bits to 0 and keep corner bits as 1.

2) Answer is XOR of generated number and original number.

C++

```
 // CPP program to toggle first and last
// bits of a number
#include <iostream>
using namespace std;

// Returns a number which has same bit
```

```
// count as n and has only first and last
// bits as set.
int takeLandFsetbits(int n)
{
    // set all the bit of the number
    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    // Adding one to n now unsets
    // all bits and moves MSB to
    // one place. Now we shift
    // the number by 1 and add 1.
    return ((n + 1) >> 1) + 1;
}

int toggleFandLbits(int n)
{
    // if number is 1
    if (n == 1)
        return 0;

    // take XOR with first and
    // last set bit number
    return n ^ takeLandFsetbits(n);
}

// Driver code
int main()
{
    int n = 10;
    cout << toggleFandLbits(n);
    return 0;
}
```

Java

```
 // Java program to toggle first and last
// bits of a number
import java.io.*;

class GFG {

    // Returns a number which has same bit
    // count as n and has only first and last
    // bits as set.
```

```
    static int takeLandFsetbits(int n)
    {
        // set all the bit of the number
        n |= n >> 1;
        n |= n >> 2;
        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;

        // Adding one to n now unsets
        // all bits and moves MSB to
        // one place. Now we shift
        // the number by 1 and add 1.
        return ((n + 1) >> 1) + 1;
    }

    static int toggleFandLbits(int n)
    {
        // if number is 1
        if (n == 1)
            return 0;

        // take XOR with first and
        // last set bit number
        return n ^ takeLandFsetbits(n);
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 10;
        System.out.println(toggleFandLbits(n));
    }
}

/*This code is contributed by Nikita Tiwari.*/
```

Python3

```
 # Python 3 program to toggle first
#  and last bits of a number.


# Returns a number which has same bit
# count as n and has only first and last
# bits as set.
def takeLandFsetbits(n) :
```

```
# set all the bit of the number
    n = n | n >> 1
    n = n | n >> 2
    n = n | n >> 4
    n = n | n >> 8
    n = n | n >> 16

    # Adding one to n now unsets
    # all bits and moves MSB to
    # one place. Now we shift
    # the number by 1 and add 1.
    return ((n + 1) >> 1) + 1

def toggleFandLbits(n) :
    # if number is 1
    if (n == 1) :
        return 0

    # take XOR with first and
    # last set bit number
    return n ^ takeLandFsetbits(n)

# Driver code
n = 10
print(toggleFandLbits(n))

# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# program to toggle first and last
// bits of a number
using System;

class GFG {

    // Returns a number which has same bit
    // count as n and has only first and last
    // bits as set.
    static int takeLandFsetbits(int n)
    {
        // set all the bit of the number
        n |= n >> 1;
        n |= n >> 2;
        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;
```

```
        // Adding one to n now unsets
        // all bits and moves MSB to
        // one place. Now we shift
        // the number by 1 and add 1.
        return ((n + 1) >> 1) + 1;
    }

    static int toggleFandLbits(int n)
    {

        // if number is 1
        if (n == 1)
            return 0;

        // take XOR with first and
        // last set bit number
        return n ^ takeLandFsetbits(n);
    }

    // Driver code
    public static void Main()
    {

        int n = 10;

        Console.WriteLine(toggleFandLbits(n));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program to toggle first and last
// bits of a number

// Returns a number which has same bit
// count as n and has only first and last
// bits as set.
function takeLandFsetbits($n)
{
    // set all the bit of the number
    $n |= $n >> 1;
    $n |= $n >> 2;
    $n |= $n >> 4;
    $n |= $n >> 8;
    $n |= $n >> 16;
```

```
    // Adding one to n now unsets
    // all bits and moves MSB to
    // one place. Now we shift
    // the number by 1 and add 1.
    return (($n + 1) >> 1) + 1;
}

function toggleFandLbits(int $n)
{
    // if number is 1
    if ($n == 1)
        return 0;

    // take XOR with first and
    // last set bit number
    return $n ^ takeLandFsetbits($n);
}

// Driver code
$n = 10;
echo toggleFandLbits($n);

// This code is contributed by mits
?>
```

Output :

```
3
```

Time Complexity is O(1).

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/toggle-first-last-bits-number/

Chapter 326

Toggle the last m bits

Toggle the last m bits - GeeksforGeeks

Given a non-negative number **n**. The problem is to toggle the last **m** bits in the binary representation of **n**. A **toggle** operation flips a bit **0** to **1** and a bit **1** to **0**.

Constraint: 1 <= m <= n.

Examples:

```
Input : n = 21, m = 2
Output : 22
(21)10 = (10101)2
(22)10 = (10110)2
The last two bits in the binary
representation of 21 are toggled.

Input : n = 107, m = 4
Output : 100
```

Approach: Following are the steps:

1. Calculate **num** = (1 << m) – 1. This will produce a number **num** having **m** number of bits and all will be set.
2. Now, perform **n = n ^ num**. This will toggle the last **m** bits in **n**.

C++

```
 // C++ implementation to
// toggle the last m bits
#include <bits/stdc++.h>
using namespace std;
```

```
// function to toggle
// the last m bits
unsigned int toggleLastMBits
         (unsigned int n, unsigned int m)
{

    // calculating a number
    // 'num' having 'm' bits
    // and all are set.
    unsigned int num = (1 << m) - 1;

    // toggle the last m bits
    // and return the number
    return (n ^ num);
}

// Driver code
int main()
{
    unsigned int n = 107;
    unsigned int m = 4;
    cout << toggleLastMBits(n, m);
    return 0;
}
```

Java

```
 // Java implementation to
// toggle the last m bits
import java.util.*;
import java.lang.*;

public class GfG{

    // function to toggle
    // the last m bits
    public static int toggleLastMBits
                      (int n, int m)
    {

        // calculating a number
        // 'num' having 'm' bits
        // and all are set
        int num = (1 << m) - 1;

        // toggle the last m bits
        // and return the number
```

```
        return (n ^ num);
    }

    // Driver function
    public static void main(String argc[]){
        int n = 107;
        int m = 4;
        n =  toggleLastMBits(n, m);
        System.out.println(n);

    }
}

// This code is contributed by Sagar Shukla.
```

Python3

```
 # Python implementation to
# toggle the last m bits

# function to toggle
# the last m bits
def toggleLastMBits(n,m):

    # calculating a number
    # 'num' having 'm' bits
    # and all are set.
    num = (1 << m) - 1

    # toggle the last m bits
    # and return the number
    return (n ^ num)

# Driver code

n = 107
m = 4
print(toggleLastMBits(n, m))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# implementation to
// toggle the last m bits
using System;
```

```
namespace Toggle
{
    public class GFG
    {

    // Function to toggle the last m bits
    public static int toggleLastMBits(int n, int m)
    {

        // Calculating a number 'num' having
        // 'm' bits and all are set
        int num = (1 << m) - 1;

        // Toggle the last m bits
        // and return the number
        return (n ^ num);
    }

    // Driver Code
    public static void Main() {

        int n = 107, m = 4;
        n = toggleLastMBits(n, m);
        Console.Write(n);

    }
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP implementation to
// toggle the last m bits

    // function to toggle
    // the last m bits
    function toggleLastMBits($n, $m)
    {

        // calculating a number
        // 'num' having 'm' bits
        // and all are set.
        $num = (1 << $m) - 1;
```

```
        // toggle the last m bits
        // and return the number
        return ($n ^ $num);
    }

// Driver code
{
    $n = 107;
    $m = 4;
    echo toggleLastMBits($n, $m);
    return 0;
}

// This code is contributed by nitin mittal.
?>
```

Output:

```
100
```

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/toggle-last-m-bits/

Chapter 327

Toggling k-th bit of a number

Toggling k-th bit of a number - GeeksforGeeks

For a given number n, if k-th bit is 0, then toggle it to 1 and if it is 1 then, toggle it to 0.

Examples :

```
Input : n = 5, k = 1
Output : 4
5 is represented as 101 in binary
and has its first bit 1, so toggling
it will result in 100 i.e. 4.

Input : n = 2, k = 3
Output : 6

Input : n = 75, k = 4
Output : 67
```

Below are simple steps to find value of k-th bit

```
1) Left shift given number 1 by k-1 to create
   a number that has only set bit as k-th bit.
    temp = 1 << (k-1)
2) Return bitwise XOR of temp and n.  Since temp
   has only k-th bit set, doing XOR would toggle
   only this bit.
```

Example :

```
 n = 75 and k = 4
 temp = 1 << (k-1) = 1 << 3 = 8
 Binary Representation of temp = 0..00001000
 Binary Representation of n = 0..01001011
 Bitwise XOR of two numbers = 0..01000011
```

C++

```
 // CPP program to toggle k-th bit of n
#include<iostream>
using namespace std;

int toggleKthBit(int n, int k)
{
    return (n ^ (1 << (k-1)));
}

// Driver code
int main()
{
    int n = 5, k = 1;
    cout << toggleKthBit(n , k);
    return 0;
}
```

Java

```
// Java program to toogle
// k-th bit of a number

class Toggle
{
    static int toggleKthBit(int n, int k)
    {
        return (n ^ (1 << (k-1)));
    }

    // main function
    public static void main (String[] args)
    {
        int n = 5, k = 1;
        System.out.println(toggleKthBit(n , k));
    }
}
```

Python3

```
 # Python3 code to toggle k-th bit of n

def toggleKthBit(n, k):
    return (n ^ (1 << (k-1)))

# Driver code
n = 5
k = 1
print( toggleKthBit(n , k))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# program to toogle
// k-th bit of a number
using System;

class GFG {

    static int toggleKthBit(int n, int k)
    {
        return (n ^ (1 << (k-1)));
    }

    // main function
    public static void Main()
    {
        int n = 5, k = 1;

        Console.WriteLine(toggleKthBit(n , k));
    }
}

//This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// Php program to toggle k-th bit of n

function toggleKthBit($n, $k)
{
    return ($n ^ (1 << ($k - 1)));
}

// Driver code
```

```
$n = 5;
$k = 1;
echo toggleKthBit($n, $k);

// This code is contributed by Ajit
?>
```

Output :

```
4
```

Improved By : jit_t

Source

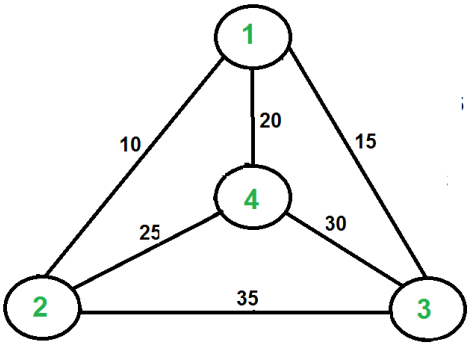
https://www.geeksforgeeks.org/toggling-k-th-bit-number/

Chapter 328

Travelling Salesman Problem | Set 1 (Naive and Dynamic Programming)

Travelling Salesman Problem | Set 1 (Naive and Dynamic Programming) - GeeksforGeeks

Travelling Salesman Problem (TSP): Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.
Note the difference between Hamiltonian Cycle and TSP. The Hamiltoninan cycle problem is to find if there exist a tour that visits every city exactly once. Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.



For example, consider the graph shown in figure on right side. A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80.

The problem is a famous NP hardproblem. There is no polynomial time know solution for this problem.

Following are different solutions for the traveling salesman problem.

Naive Solution:
1) Consider city 1 as the starting and ending point.
2) Generate all (n-1)! Permutationsof cities.
3) Calculate cost of every permutation and keep track of minimum cost permutation.
4) Return the permutation with minimum cost.

Time Complexity: $\Theta(n!)$

Dynamic Programming:
Let the given set of vertices be {1, 2, 3, 4,....n}. Let us consider 1 as starting and ending point of output. For every other vertex i (other than 1), we find the minimum cost path with 1 as the starting point, i as the ending point and all vertices appearing exactly once. Let the cost of this path be cost(i), the cost of corresponding Cycle would be cost(i) + dist(i, 1) where dist(i, 1) is the distance from i to 1. Finally, we return the minimum of all [cost(i) + dist(i, 1)] values. This looks simple so far. Now the question is how to get cost(i)?
To calculate cost(i) using Dynamic Programming, we need to have some recursive relation in terms of sub-problems. Let us define a term *C(S, i) be the cost of the minimum cost path visiting each vertex in set S exactly once, starting at 1 and ending at i.*
We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset, then we calculate C(S, i) for all subsets S of size 3 and so on. Note that 1 must be present in every subset.

```
If size of S is 2, then S must be {1, i},
 C(S, i) = dist(1, i)
Else if size of S is greater than 2.
 C(S, i) = min { C(S-{i}, j) + dis(j, i)} where j belongs to S, j != i and j != 1.
```

For a set of size n, we consider n-2 subsets each of size n-1 such that all subsets don't have nth in them.
Using the above recurrence relation, we can write dynamic programming based solution. There are at most $O(n*2^n)$ subproblems, and each one takes linear time to solve. The total running time is therefore $O(n^2*2^n)$. The time complexity is much less than O(n!), but still exponential. Space required is also exponential. So this approach is also infeasible even for slightly higher number of vertices.

We will soon be discussing approximate algorithms for travelling salesman problem.

Next Article: Traveling Salesman Problem | Set 2

References:
http://www.lsi.upc.edu/~mjserna/docencia/algofib/P07/dynprog.pdf
http://www.cs.berkeley.edu/~vazirani/algorithms/chap6.pdf

Source

https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/

Chapter 329

Turn off the rightmost set bit

Turn off the rightmost set bit - GeeksforGeeks

Write a program that unsets the rightmost set bit of an integer.

Examples :

```
Input:  12 (00...01100)
Output: 8 (00...01000)

Input:  7 (00...00111)
Output: 6 (00...00110)
```

Let the input number be n. n-1 would have all the bits flipped after the rightmost set bit (including the set bit). So, doing n&(n-1) would give us the required result.

C

```
 #include<stdio.h>

// unsets the rightmost set bit
// of n and returns the result
int fun(unsigned int n)
{
return n & (n - 1);
}

// Driver Code
int main()
{
int n = 7;
printf("The number after unsetting the");
```

```
printf(" rightmost set bit %d", fun(n));

getchar();
return 0;
}
```

Java

```
 // Java program to unset the
// rightmost set bit of an integer.

class GFG {

/* unsets the rightmost set bit
of n and returns the result */
static int fun(int n)
{
    return n & (n - 1);
}

// Driver code
public static void main(String arg[])
{
    int n = 7;
    System.out.print("The number after unsetting " +
                  "the rightmost set bit " + fun(n));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # unsets the rightmost set bit
# of n and returns the result
def fun(n):

     return n & (n-1)

# Driver code

n = 7
print("The number after unsetting the rightmost set bit" , fun(n))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to unset the
// rightmost set bit of an integer.
using System;

class GFG
{

    /* unsets the rightmost set bit
    of n and returns the result */
    static int fun(int n)
    {
        return n & (n - 1);
    }

    // Driver code
    public static void Main()
    {
    int n = 7;
    Console.Write("The number after unsetting " +
                "the rightmost set bit " + fun(n));
    }

}

// This code is contributed by Sam007
```

PHP

```
 <?php
// unsets the rightmost set bit
// of n and returns the result
function fun($n)
{
return $n & ($n - 1);
}

// Driver Code
$n = 7;
echo "The number after unsetting the".
     " rightmost set bit ", fun($n);

// This code is contributed by vt_m.

?>
```

Output :

```
The number after unsetting the rightmost set bit 6
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/turn-off-the-rightmost-set-bit/

Chapter 330

Two odd occurring elements in an array where all other occur even times

Two odd occurring elements in an array where all other occur even times - GeeksforGeeks

Given an array where all elements appear even number of times except two, print the two odd occurring elements. It may be assumed that the size of array is at-least two.

Examples:

```
Input : arr[] = {2, 3, 8, 4, 4, 3, 7, 8}
Output : 2 7

Input : arr[] = {15, 10, 10, 50 7, 5, 5, 50, 50, 50, 50, 50}
Output : 7 15
```

A **simple solution** is to use two nested loops. The outer loop traverses through all elements. The inner loop counts occurrences of the current element. We print the elements whose counts of occurrences are odd. Time complexity if this solution is $O(n^2)$

A **better solution** is to use hashing. Time complexity of this solution is O(n) but it requires extra space.

An **efficient solution** is to use bitwise operators. The idea is based on approach used in two missing elements and two repeating elements.

C++

```
 // CPP code to find two odd occurring elements
// in an array where all other elements appear
```

```
// even number of times.
#include <bits/stdc++.h>
using namespace std;

void printOdds(int arr[], int n)
{
    // Find XOR of all numbers
    int res = 0;
    for (int i = 0; i < n; i++)
        res = res ^ arr[i];

    // Find a set bit in the XOR (We find
    // rightmost set bit here)
    int set_bit = res & (~(res - 1));

    // Traverse through all numbers and
    // divide them in two groups
    // (i) Having set bit set at same
    //     position as the only set bit
    //     in set_bit
    // (ii) Having 0 bit at same position
    //      as the only set bit in set_bit
    int x = 0, y = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i] & set_bit)
            x = x ^ arr[i];
        else
            y = y ^ arr[i];
    }

    // XOR of two different sets are our
    // required numbers.
    cout << x << " " << y;
}

// Driver code
int main()
{
    int arr[] = { 2, 3, 3, 4, 4, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printOdds(arr, n);
    return 0;
}
```

Java

```
 // Java code to find two
// odd occurring elements
```

```
// in an array where all
// other elements appear
// even number of times.

class GFG
{
static void printOdds(int arr[],
                      int n)
{
    // Find XOR of
    // all numbers
    int res = 0;
    for (int i = 0; i < n; i++)
        res = res ^ arr[i];

    // Find a set bit in the
    // XOR (We find rightmost
    // set bit here)
    int set_bit = res &
                  (~(res - 1));

    // Traverse through all
    // numbers and divide them
    // in two groups (i) Having
    // set bit set at same position
    // as the only set bit in
    // set_bit (ii) Having 0 bit at
    // same position as the only
    // set bit in set_bit
    int x = 0, y = 0;
    for (int i = 0; i < n; i++)
    {
        if ((arr[i] & set_bit) != 0)
            x = x ^ arr[i];
        else
            y = y ^ arr[i];
    }

    // XOR of two different
    // sets are our required
    // numbers.
    System.out.println( x + " " + y);
}

// Driver code
public static void main(String [] args)
{
    int arr[] = { 2, 3, 3,
```

```
                4, 4, 5 };
    int n = arr.length;
    printOdds(arr, n);
}
}

// This code is contributed by
// Smitha Dinesh Semwal
```

Python3

```
 # Python 3 code to find two
# odd occurring elements in
# an array where all other
# elements appear even number
# of times.
def printOdds(arr, n):

    # Find XOR of all numbers
    res = 0
    for i in range(0, n):
        res = res ^ arr[i]

    # Find a set bit in
    # the XOR (We find
    # rightmost set bit here)
    set_bit = res & (~(res - 1))

    # Traverse through all numbers
    # and divide them in two groups
    # (i) Having set bit set at
    # same position as the only set
    # bit in set_bit
    # (ii) Having 0 bit at same
    # position as the only set
    # bit in set_bit
    x = 0
    y = 0
    for i in range(0, n):
        if (arr[i] & set_bit):
            x = x ^ arr[i]
        else:
            y = y ^ arr[i]

    # XOR of two different
    # sets are our
    # required numbers.
    print(x , y, end = "")
```

```
# Driver code
arr = [2, 3, 3, 4, 4, 5 ]
n = len(arr)
printOdds(arr, n)

# This code is contributed
# by Smitha
```

C#

```
 // C# code to find two
// odd occurring elements
// in an array where all
// other elements appear
// even number of times.
using System;

class GFG
{
static void printOdds(int []arr,
                      int n)
{
    // Find XOR of
    // all numbers
    int res = 0;
    for (int i = 0; i < n; i++)
        res = res ^ arr[i];

    // Find a set bit in the
    // XOR (We find rightmost
    // set bit here)
    int set_bit = res &
               (~(res - 1));

    // Traverse through all
    // numbers and divide them
    // in two groups (i) Having
    // set bit set at same position
    // as the only set bit in
    // set_bit (ii) Having 0 bit at
    // same position as the only
    // set bit in set_bit
    int x = 0, y = 0;
    for (int i = 0; i < n; i++)
    {
        if ((arr[i] & set_bit) != 0)
            x = x ^ arr[i];
```

```
        else
            y = y ^ arr[i];
    }

    // XOR of two different
    // sets are our required
    // numbers.
    Console.WriteLine(x + " " + y);
}

// Driver code
public static void Main()
{
    int []arr = { 2, 3, 3,
                  4, 4, 5 };
    int n = arr.Length;
    printOdds(arr, n);
}
}

// This code is contributed by
// Akanksha Rai(Abby_akku)
```

PHP

Output:

```
5 2
```

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

Improved By : Smitha Dinesh Semwal, Abby_akku

Source

https://www.geeksforgeeks.org/two-odd-occurring-elements-array-occur-even-times/

Chapter 331

Unique element in an array where all elements occur k times except one

Unique element in an array where all elements occur k times except one - GeeksforGeeks

Given an array which contains all elements occurring k times, but one occurs only once. Find that unique element.

Examples:

```
Input  : arr[] = {6, 2, 5, 2, 2, 6, 6}
           k = 3
Output : 5
Every element appears 3 times accept 5.

Input  : arr[] = {2, 2, 2, 10, 2}
           k = 4
Output : 10
Every element appears 4 times accept 10.
```

A **Simple Solution** is to use two nested loops. The outer loop picks an element one by one starting from the leftmost element. The inner loop checks if the element is present k times or not. If present, then ignores the element, else prints the element.

Time Complexity of above solution is $O(n^2)$. We can **Use Sorting** to solve the problem in O(nLogn) time. The idea is simple, first sort the array so that all occurrences of every element become consecutive. Once the occurrences become consecutive, we can traverse the sorted array and print the unique element in O(n) time.

We can **Use Hashing** to solve this in O(n) time on average. The idea is to traverse the given array from left to right and keep track of visited elements in a hash table. Finally print the element with count 1.

The hashing based solution requires O(n) extra space. We can **use bitwise AND** to find the unique element in O(n) time and constant extra space.

1. Create an array **count**[] of size equal to number of bits in binary representations of numbers.
2. Fill count array such that count[i] stores count of array elements with i-th bit set.
3. Form result using count array. We put 1 at a position i in result if count[i] is not multiple of k. Else we put 0.

```
 // CPP program to find unique element where
// every element appears k times except one
#include <bits/stdc++.h>
using namespace std;

int findUnique(unsigned int a[], int n, int k)
{
    // Create a count array to store count of
    // numbers that have a particular bit set.
    // count[i] stores count of array elements
    // with i-th bit set.
    int INT_SIZE = 8 * sizeof(unsigned int);
    int count[INT_SIZE];
    memset(count, 0, sizeof(count));

    // AND(bitwise) each element of the array
    // with each set digit (one at a time)
    // to get the count of set bits at each
    // position
    for (int i = 0; i < INT_SIZE; i++)
        for (int j = 0; j < n; j++)
            if ((a[j] & (1 << i)) != 0)
                count[i] += 1;

    // Now consider all bits whose count is
    // not multiple of k to form the required
    // number.
    unsigned res = 0;
    for (int i = 0; i < INT_SIZE; i++)
        res += (count[i] % k) * (1 << i);
    return res;
}

// Driver Code
int main()
{
    unsigned int a[] = { 6, 2, 5, 2, 2, 6, 6 };
    int n = sizeof(a) / sizeof(a[0]);
    int k = 3;
```

```
    cout << findUnique(a, n, k);
    return 0;
}
```

Output:

```
5
```

Source

https://www.geeksforgeeks.org/find-unique-element-element-occurs-k-times-except-one/

Chapter 332

Unset bits in the given range

Unset bits in the given range - GeeksforGeeks

Given a non-negative number **n** and two values **l** and **r**. The problem is to unset the bits in the range **l** to **r** in the binary representation of **n**, i.e, to unset bits from the rightmost **lth** bit to the rightmost **rth** bit.

Constraint: 1 <= l <= r <= number of bits in the binary representation of **n**.

Examples:

```
Input : n = 42, l = 2, r = 5
Output : 32
(42)10 = (101010)2
(32)10 = (100000)2
The bits in the range 2 to 5 in the binary
representation of 42 have been unset.

Input : n = 63, l = 1, r = 4
Output : 48
```

Approach: Following are the steps:

1. Calculate **num** = (1 << (sizeof(int) * 8 – 1)) – 1. This will produce the highest positive integer **num**. All the bits in **num** will be set.
2. Toggle bits in the range **l** to **r** in **num**. Refer this post.
3. Now, perform **n = n & num**. This will unset the bits in the range **l** to **r** in **n**.
4. Return **n**.

Note: The **sizeof(int)** has been used as input is of **int** data type. For large inputs you can use **long int** or **long long int** datatypes in place of **int**.

C/C++

```
 // C++ implementation to unset bits in the given range
#include <bits/stdc++.h>

using namespace std;

// function to toggle bits in the given range
unsigned int toggleBitsFromLToR(unsigned int n,
                                unsigned int l,
                                unsigned int r)
{
    // calculating a number 'num' having 'r' number of bits
    // and bits in the range l to r are the only set bits
    int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

    // toggle the bits in the range l to r in 'n'
    // and return the number
    return (n ^ num);
}

// function to unset bits in the given range
unsigned int unsetBitsInGivenRange(unsigned int n,
                                   unsigned int l, unsigned int r)
{
    // 'num' is the highest positive integer number
    // all the bits of 'num' are set
    unsigned int num = (1 << (sizeof(int) * 8 - 1)) - 1;

    // toggle the bits in the range l to r in 'num'
    num = toggleBitsFromLToR(num, l, r);

    // unset the bits in the range l to r in 'n'
    // and return the number
    return (n & num);
}

// Driver program to test above
int main()
{
    unsigned int n = 42;
    unsigned int l = 2, r = 5;
    cout << unsetBitsInGivenRange(n, l, r);
    return 0;
}
```

Java

```
 // Java implementation to unset bits in the given range
import java.io.*;
```

```
class GFG
{
    // Function to toggle bits in the given range
    static int toggleBitsFromLToR(int n, int l, int r)
    {
        // calculating a number 'num' having 'r' number of bits
        // and bits in the range l to r are the only set bits
        int num = ((1 << r) - 1) ^ ((1 << (l - 1)) - 1);

        // toggle the bits in the range l to r in 'n'
        // and return the number
        return (n ^ num);
    }

    // Function to unset bits in the given range
    static int unsetBitsInGivenRange(int n, int l, int r)
    {
        // 'num' is the highest positive integer number
        // all the bits of 'num' are set
        int num = (1 << (4 * 8 - 1)) - 1;

        // toggle the bits in the range l to r in 'num'
        num = toggleBitsFromLToR(num, l, r);

        // unset the bits in the range l to r in 'n'
        // and return the number
        return (n & num);
    }

    // driver program
    public static void main (String[] args)
    {
        int n = 42;
        int l = 2, r = 5;
        System.out.println(unsetBitsInGivenRange(n, l, r));
    }
}

// Contributed by Pramod Kumar
```

Python3

```
 # python implementation to unset bits
# in the given range

# Function to toggle bits in the
# given range
```

```
def toggleBitsFromLToR(n, l, r):

    # calculating a number 'num'
    # having 'r' number of bits
    # and bits in the range l to
    # r are the only set bits
    num = (((1 << r) - 1) ^
           ((1 << (l - 1)) - 1))

    # toggle the bits in the range
    # l to r in 'n' and return the
    # number
    return (n ^ num)

# Function to unset bits in the
# given range
def unsetBitsInGivenRange(n, l, r):

    # 'num' is the highest positive
    # integer number all the bits
    # of 'num' are set
    num = (1 << (4 * 8 - 1)) - 1

    # toggle the bits in the range
    # l to r in 'num'
    num = toggleBitsFromLToR(num, l, r)

    # unset the bits in the range
    # l to r in 'n' and return the
    # number
    return (n & num)

# Driver code
n = 42
l = 2
r = 5
print(unsetBitsInGivenRange(n, l, r))

# This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP implementation to unset
// bits in the given range

    // Function to toggle bits
    // in the given range
```

```
    function toggleBitsFromLToR($n, $l, $r)
    {

        // calculating a number 'num'
        // having 'r' number of bits
        // and bits in the range l to
        // r are the only set bits
        $num = ((1 << $r) - 1) ^
               ((1 << ($l - 1)) - 1);

        // toggle the bits in the
        // range l to r in 'n'
        // and return the number
        return ($n ^ $num);
    }

    // Function to unset bits
    // in the given range
    function unsetBitsInGivenRange($n,$l,$r)
    {

        // 'num' is the highest
        // positive integer number
        // all the bits of 'num' are set
        $num = (1 << (4 * 8 - 1)) - 1;

        // toggle the bits in the
        // range l to r in 'num'
        $num = toggleBitsFromLToR($num, $l, $r);

        // unset the bits in the
        // range l to r in 'n'
        // and return the number
        return ($n & $num);
    }

        // Driver Code
        $n = 42;
        $l = 2;
        $r = 5;
        echo unsetBitsInGivenRange($n, $l, $r);

// This code is contributed by Sam007
?>
```

Output:

```
32
```

Improved By : Sam007

Source

https://www.geeksforgeeks.org/unset-bits-given-range/

Chapter 333

Unset the last m bits

Unset the last m bits - GeeksforGeeks

Given a non-negative number **n**. The problem is to unset the last **m** bits in the binary representation of **n**.

Constraint: 1 <= m <= num, where **num** is the number of bits in the binary representation of **n**.

Examples:

```
Input : n = 10, m = 2
Output : 8
(10)10 = (1010)2
(8)10 = (1000)2
The last two bits in the binary
representation of 10 have been unset.

Input : n = 150, m = 4
Output : 144
```

Approach: Following are the steps:

1. Calculate **num** = (1 << (sizeof(int) * 8 – 1)) – 1. This will produce the highest positive integer **num**. All the bits in **num** will be set.
2. Toggle the last **m** bits in **num**. Refer this post.
3. Now, perform **n = n & num**. This will unset the last **m** bits in **n**.
4. Return **n**.

Note: The **sizeof(int)** has been used as input is of **int** data type. For large inputs you can use **long int** or **long long int** datatypes in place of **int**.

C++

```
 // C++ implementation to unset bits the last m bits
#include <bits/stdc++.h>
using namespace std;

// function to toggle the last m bits
unsigned int toggleLastMBits(unsigned int n,
                             unsigned int m)
{
    // calculating a number 'num' having 'm' bits
    // and all are set
    unsigned int num = (1 << m) - 1;

    // toggle the last m bits and return the number
    return (n ^ num);
}

// function to unset bits the last m bits
unsigned int unsetLastMBits(unsigned int n,
                            unsigned int m)
{
    // 'num' is the highest positive integer number
    // all the bits of 'num' are set
    unsigned int num = (1 << (sizeof(int) * 8 - 1)) - 1;

    // toggle the last 'm' bits in 'num'
    num = toggleLastMBits(num, m);

    // unset the last 'm' bits in 'n'
    // and return the number
    return (n & num);
}

// Driver program to test above
int main()
{
    unsigned int n = 150, m = 4;
    cout << unsetLastMBits(n, m);
    return 0;
}
```

Python3

```
 # Python3 implementation to unset
# bits the last m bits
import sys

# function to toggle the last m bits
def toggleLastMBits (n, m):
```

```
    # calculating a number 'num'
    # having 'm' bits and all are set
    num = (1 << m) - 1

    # toggle the last m bits
    # and return the number
    return (n ^ num)

# function to unset bits
# the last m bits
def unsetLastMBits(n, m):

    # 'num' is the highest positive integer
    # number all the bits of 'num' are set
    num = (1 << (sys.getsizeof(int) * 8 - 1)) - 1

    # toggle the last 'm' bits in 'num'
    num = toggleLastMBits(num, m)

    # unset the last 'm' bits in 'n'
    # and return the number
    return (n & num)

# Driven code
n = 150
m = 4
print (unsetLastMBits(n, m))

# This code is contributed by "rishabh_jain".
```

PHP

```
 <?php
// php implementation to unset
// bits the last m bits

// function to toggle
// the last m bits
function toggleLastMBits($n, $m)
{

    // calculating a number 'num'
    // having 'm' bits
    // and all are set
    $num = (1 << $m) - 1;

    // toggle the last m bits
```

```
    // and return the number
    return ($n ^ $num);
}

// function to unset bits
// the last m bits
function unsetLastMBits($n,$m)
{

    // 'num' is the highest positive
    // integer number all the bits
    // of 'num' are set

    //4 is Size of integer 32 bit
    $num = (1 << (4 * 8 - 1)) - 1;

    // toggle the last 'm' bits in 'num'
    $num = toggleLastMBits($num, $m);

    // unset the last 'm' bits in 'n'
    // and return the number
    return ($n & $num);
}

    // Drivercode
    $n = 150;
    $m = 4;
    echo unsetLastMBits($n, $m);

// This code is contributed by mits
?>
```

Output:

```
144
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/unset-last-m-bits/

Chapter 334

Value in a given range with maximum XOR

Value in a given range with maximum XOR - GeeksforGeeks

Given positive integers N, L, and R, we have to find the maximum value of N X, where X [L, R].

Examples:

> **Input :** N = 7
> L = 2
> R = 23
> **Output :** 23
> Explanation : When X = 16, we get 7 16 = 23 which is the maximum value for all X [2, 23].
>
> **Input :** N = 10
> L = 5
> R = 12
> **Output :** 15
> Explanation : When X = 5, we get 10 5 = 15 which is the maximum value for all X [5, 12].

Brute force approach: We can solve this problem using brute force approach by looping over all integers over the range [L, R] and taking their XOR with N, while keeping a record of the maximum result encountered so far. The complexity of this algorithm will be O(R – L), and it is not feasible when the input variables approach high values such as 10^9.

Efficient approach: Since the XOR of two bits is 1 if and only if they are complementary to each other, we need X to have complementary bits to that of N to have the maximum value. We will iterate from the largest bit ($\log_2(R)^{th}$ bit) to the lowest (0^{th} bit). The following two cases can arise for each bit:

1. If the bit is not set, i.e. 0, we will try to set it in X. If setting this bit to 1 results in X exceeding R, then we will not set it.
2. If the bit is set, i.e. 1, then we will try to unset it in X. If the current value of X is already greater than or equal to L, then we can safely unset the bit. In the other case, we will check if setting all of the next sits is enough to keep X >= L. If not, then we are required to set the current bit. Observe that setting all of the next bits is equivalent to adding $(1 << b) - 1$, where b is the current bit.

The time complexity of this approach is $O(\log_2(R))$.

C++

```
 // CPP program to find the x in range [l, r]
// such that x ^ n is maximum.
#include <cmath>
#include <iostream>
using namespace std;

// Function to calculate the maximum value of
// N ^ X, where X is in the range [L, R]
int maximumXOR(int n, int l, int r)
{
    int x = 0;
    for (int i = log2(r); i >= 0; --i) {
        if (n & (1 << i)) { // Set bit
            if ((x > r) || (x + (1 << i) - 1 < l))
                x ^= (1 << i);
        }
        else { // Unset bit
            if ((x ^ (1 << i)) <= r)
                x ^= (1 << i);
        }
    }
    return n ^ x;
}

// Driver function
int main()
{
    int n = 7, l = 2, r = 23;
    cout << "The output is " << maximumXOR(n, l, r);
    return 0;
}
```

Java

```
 // Java program to find the x in range [l, r]
// such that x ^ n is maximum.
```

```
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{
// Function to calculate the maximum value of
// N ^ X, where X is in the range [L, R]
static int maximumXOR(int n, int l, int r)
{
    int x = 0;
    for (int i = (int)(Math.log(r)/Math.log(2)); i >= 0; --i)
    {
        if ((n & (1 << i))>0) // Set bit
        {
            if ((x > r) || (x + (1 << i) - 1 < l))
                x ^= (1 << i);
        }
        else // Unset bit
        {
            if ((x ^ (1 << i)) <= r)
                x ^= (1 << i);
        }
    }
    return n ^ x;
}

// Driver function
public static void main(String args[])
{
    int n = 7, l = 2, r = 23;
    System.out.println( "The output is " + maximumXOR(n, l, r));

}
}

// This code is Contributed by tufan_gupta2000
```

Output:

```
The output is 23
```

Improved By : tufan_gupta2000

Source

https://www.geeksforgeeks.org/value-in-a-given-range-with-maximum-xor/

Chapter 335

Variation in Nim Game

Variation in Nim Game - GeeksforGeeks

Prerequisites:
Sprague Gruncy theorem
Grundy Numbers

Nim is a famous game in which two players take turns removing items from distinct piles. During each turn, a player must remove one or more items from a single, non-empty pile. The winner of the game is whichever player removes the last item from the last non-empty pile.
Now, For each non-empty pile, either player can remove zero items from that pile and have it count as their move; however, this move can only be performed once per pile by either player.
Given the number of items in each pile, determine who will win the game; Player 1 or player 2. If player 1 starts the game and both plays optimally.

Examples:

```
Input : 3 [18, 47, 34]
Output : Player 2 wins
G = g(18)^g(47)^g(34) = (17)^(48)^(33) = 0
Grundy number(G), for this game is zero.
Player 2 wins.

Input : 3 [32, 49, 58]
Output : Player 1 wins
G = g(31)^g(50)^g(57) = (17)^(48)^(33) = 20
Grundy number(G), for this game is non-zero.
Player 1 wins.
```

Approach:
Grundy number for each pile is calculated based on the number of stones.To compensate

the zero move we will have to modify grundy values we used in standard nim game.
If pile size is odd; grundy number is size+1 and
if pile size is even; grundy number is size-1.
We XOR all the grundy number values to check if final Grundy number(G) of game is non zero or not to decide who is winner of game.

Explanation:
Grundy number of a state is the **smallest positive integer that cannot be reached in one valid move**.
So, we need to calculate **mex value** for each n, bottom up wise so that we can induce the grundy number for each n. where n is the pile size and valid move is the move that will lead the current player to winning state.

Winning state: A tuple of values from where the current player will win the game no matter what opponent does. (If G!=0)
Losing state: A tuple of values from where the current player will loose the game no matter what opponent does. (If G=0)

```
For a given pile size n, we have two states:
(1) n with no zero move available,
grundy number will same as standard nim game.
(2) n with zero move available, we can
reach above state and other states with
zero move remaining.

For, n = 0, g(0) = 0, empty pile

For, n = 1, we can reach two states:
(1) n = 0 (zero move not used)
(2) n = 1 (zero move used)
Therefore, g(1) = mex{0, 1} which implies
that g(1)=2.

For, n = 2, we can reach :
(1) n = 0 (zero move not used) state
because this is a valid move.
(2) n = 1 is not a valid move, as it will
lead the current player into loosing state.
Therefore, g(2) = mex{0} which implies
that g(2)=1.

If we try to build a solution bottom-up
like this, it turns out that if n is even,
the grundy number is n - 1 and when it is odd,
the grundy is n + 1.
```

Below is the implementation of above approach:

C++

```
 // CPP program for the variation
// in nim game
#include <bits/stdc++.h>
using namespace std;

// Function to return final
// grundy Number(G) of game
int solve(int p[], int n)
{
    int G = 0;
    for (int i = 0; i < n; i++) {

        // if pile size is odd
        if (p[i] & 1)

            // We XOR pile size+1
            G ^= (p[i] + 1);

        else // if pile size is even

            // We XOR pile size-1
            G ^= (p[i] - 1);
    }
    return G;
}

// driver program
int main()
{
    // Game with 3 piles
    int n = 3;

    // pile with different sizes
    int p[3] = { 32, 49, 58 };

    // Function to return result of game
    int res = solve(p, n);

    if (res == 0) // if G is zero
        cout << "Player 2 wins";
    else // if G is non zero
        cout << "Player 1 wins";

    return 0;
}
```

Java

```
 // Java program for the variation
// in nim game
class GFG {

    // Function to return final
    // grundy Number(G) of game
    static int solve(int p[], int n)
    {

        int G = 0;
        for (int i = 0; i < n; i++) {

            // if pile size is odd
            if (p[i]%2!=0)

                // We XOR pile size+1
                G ^= (p[i] + 1);

            else // if pile size is even

                // We XOR pile size-1
                G ^= (p[i] - 1);
        }

        return G;
    }

    //Driver code
    public static void main (String[] args)
    {

        // Game with 3 piles
        int n = 3;

        // pile with different sizes
        int p[] = { 32, 49, 58 };

        // Function to return result of game
        int res = solve(p, n);

        if (res == 0) // if G is zero
            System.out.print("Player 2 wins");
        else // if G is non zero
            System.out.print("Player 1 wins");
    }
}
```

```
// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program for the
# variation in nim game

# Function to return final
# grundy Number(G) of game
def solve(p, n):
    G = 0
    for i in range(n):

        # if pile size is odd
        if (p[i] % 2 != 0):

            # We XOR pile size+1
            G ^= (p[i] + 1)

        # if pile size is even
        else:

            # We XOR pile size-1
            G ^= (p[i] - 1)

    return G

# Driver code

# Game with 3 piles
n = 3

# pile with different sizes
p = [32, 49, 58]

# Function to return result of game
res = solve(p, n)

if (res == 0): # if G is zero
    print("Player 2 wins")

else: # if G is non zero
    print("Player 1 wins")

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program for the variation
// in nim game
using System;
class GFG {

    // Function to return final
    // grundy Number(G) of game
    static int solve(int[] p, int n)
    {

        int G = 0;
        for (int i = 0; i < n; i++) {

            // if pile size is odd
            if (p[i] % 2 != 0)

                // We XOR pile size+1
                G ^= (p[i] + 1);

            else // if pile size is even

                // We XOR pile size-1
                G ^= (p[i] - 1);
        }

        return G;
    }

    // Driver code
    public static void Main()
    {

        // Game with 3 piles
        int n = 3;

        // pile with different sizes
        int[] p = { 32, 49, 58 };

        // Function to return result of game
        int res = solve(p, n);

        if (res == 0) // if G is zero
            Console.WriteLine("Player 2 wins");

        else // if G is non zero
            Console.WriteLine("Player 1 wins");
    }
}
```

```
// This code is contributed by vt_m.
```

PHP

```
 <?php
// php program for the variation
// in nim game

// Function to return final
// grundy Number(G) of game
function solve($p,$n)
{
    $G = 0;
    for ($i = 0; $i < $n; $i++)
    {

        // if pile size is odd
        if ($p[$i] & 1)

            // We XOR pile size+1
            $G ^= ($p[$i] + 1);

        else // if pile size is even

            // We XOR pile size-1
            $G ^= ($p[$i] - 1);
    }
    return $G;
}

    // Driver Code
    // Game with 3 piles
    $n = 3;

    // pile with different sizes
    $p= array( 32, 49, 58 );

    // Function to return result of game
    $res = solve($p, $n);

    if ($res == 0) // if G is zero
        echo "Player 2 wins";
    else // if G is non zero
        echo "Player 1 wins";

// This code is contributed by mits
?>
```

Output:

```
Player 1 wins
```

Time Complexity: O(n)

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/variation-nim-game/

Chapter 336

Ways to represent a number as a sum of 1's and 2's

Ways to represent a number as a sum of 1's and 2's - GeeksforGeeks

Given a positive integer **N**. The task is to find the number of ways of representing N as a sum of 1s and 2s.

Examples:

```
Input : N = 3
Output : 3
3 can be represented as (1+1+1), (2+1), (1+2).

Input : N = 5
Output : 8
```

For N = 1, answer is 1.
For N = 2. (1 + 1), (2), answer is 2.
For N = 3. (1 + 1 + 1), (2 + 1), (1 + 2), answer is 3.
For N = 4. (1 + 1 + 1 + 1), (2 + 1 + 1), (1 + 2 + 1), (1 + 1 + 2), (2 + 2) answer is 5.
And so on.

It can be observe that it form Fibonacci Series. So, the number of ways of representing N as a sum of 1s and 2s is $(N + 1)^{th}$ Fibonacci number.
How ?
We can easily see that the recursive function is exactly same as Fibonacci Numbers. To obtain the sum of N, we can add 1 to N – 1. Also, we can add 2 to N – 2. And only 1 and 2 are allowed to make the sum N. So, to obtain sum N using 1s and 2s, total ways are: number of ways to obtain (N – 1) + number of ways to obtain (N – 2).

We can find N'th Fibonacci Number in O(Log n) time. Please refer method 5 of this post.

Below is C++ implementation of this approach:

```
 // C++ program to find number of ways to representing
// a number as a sum of 1's and 2's
#include <bits/stdc++.h>
using namespace std;

// Function to multiply matrix.
void multiply(int F[2][2], int M[2][2])
{
    int x =  F[0][0]*M[0][0] + F[0][1]*M[1][0];
    int y =  F[0][0]*M[0][1] + F[0][1]*M[1][1];
    int z =  F[1][0]*M[0][0] + F[1][1]*M[1][0];
    int w =  F[1][0]*M[0][1] + F[1][1]*M[1][1];

    F[0][0] = x;
    F[0][1] = y;
    F[1][0] = z;
    F[1][1] = w;
}

// Power function in log n
void power(int F[2][2], int n)
{
    if( n == 0 || n == 1)
        return;
    int M[2][2] = {{1,1},{1,0}};

    power(F, n/2);
    multiply(F, F);

    if (n%2 != 0)
        multiply(F, M);
}

/* function that returns (n+1)th Fibonacci number
   Or number of ways to represent n as sum of 1's
   2's */
int countWays(int n)
{
    int F[2][2] = {{1,1},{1,0}};
    if (n == 0)
        return 0;
    power(F, n);
    return F[0][0];
}

// Driver program
int main()
{
```

```
    int n = 5;
    cout << countWays(n) << endl;
    return 0;
}
```

Output:

```
8
```

Time Complexity: O(logn).

Source

https://www.geeksforgeeks.org/ways-to-represent-a-number-as-a-sum-of-1s-and-2s/

Chapter 337

Ways to split array into two groups of same XOR value

Ways to split array into two groups of same XOR value - GeeksforGeeks

Given an array **A** of **n** integers. The task is to count the number of ways to split given array elements into two disjoint groups, such that XOR of elements of each group is equal.

Examples:

```
Input : A[] = { 1, 2, 3 }
Output : 3
{(1), (2, 3)}, {(2), (1, 3)}, {(3), (1, 2)}
are three ways with equal XOR value of two
groups.

Input :  A[] = { 5, 2, 3, 2 }
Output : 0
```

Let's denote XOR between all elements in the first group as G_1 and XOR between all elements in the second group as G_2. Now, the following relation is always correct: $G_1 \quad G_2 = A_1 \quad A_2 \quad .... \quad A_n$.
So for $G_1 = G_2$, xor between all elements of array A is equal to 0. So, in that case, answer will be $(2^n - 2)/2 = (2^{n-1} - 1)$. In second case, when XOR between all elements isn't 0, we can not split array. Answer will be 0.

C++

```
 // CPP Program to count number of ways to split
// array into two groups such that each group
// has equal XOR value
```

```
#include<bits/stdc++.h>
using namespace std;

// Return the count number of ways to split
// array into two  groups such that each group
// has equal XOR value.
int countgroup(int a[], int n)
{
  int xs = 0;
  for (int i = 0; i < n; i++)
    xs = xs ^ a[i];

  // We can split only if XOR is 0. Since
  // XOR of all is 0, we can consider all
  // subsets as one group.
  if (xs == 0)
    return (1 << (n-1)) - 1;

  return 0;
}

// Driver Program
int main()
{
  int a[] = { 1, 2, 3 };
  int n = sizeof(a)/sizeof(a[0]);
  cout << countgroup(a, n) << endl;
  return 0;
}
```

Java

```
 // Java Program to count number of ways
// to split array into two groups such
// that each group has equal XOR value
import java.io.*;
import java.util.*;

class GFG {

// Return the count number of ways to split
// array into two groups such that each group
// has equal XOR value.
static int countgroup(int a[], int n) {
    int xs = 0;
    for (int i = 0; i < n; i++)
    xs = xs ^ a[i];
```

```
    // We can split only if XOR is 0. Since
    // XOR of all is 0, we can consider all
    // subsets as one group.
    if (xs == 0)
    return (1 << (n - 1)) - 1;

    return 0;
}

// Driver program
public static void main(String args[]) {
    int a[] = {1, 2, 3};
    int n = a.length;
    System.out.println(countgroup(a, n));
}
}

// This code is contributed by Nikita Tiwari.
```

Python3

```
 # Python3 code to count number of ways
# to split array into two groups such
# that each group has equal XOR value

# Return the count of number of ways
# to split array into two groups such
# that each group has equal XOR value.
def countgroup(a, n):
    xs = 0
    for i in range(n):
        xs = xs ^ a[i]

    # We can split only if XOR is 0.
    # Since XOR of all is 0, we can
    # consider all subsets as one group.
    if xs == 0:
        return (1 << (n-1)) - 1

    return 0

# Driver Program
a = [1, 2, 3]
n = len(a)
print(countgroup(a, n))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# Program to count number of ways
// to split array into two groups such
// that each group has equal XOR value
using System;

class GFG {

    // Return the count number of ways to split
    // array into two groups such that each group
    // has equal XOR value.
    static int countgroup(int[] a, int n)
    {
        int xs = 0;
        for (int i = 0; i < n; i++)
            xs = xs ^ a[i];

        // We can split only if XOR is 0. Since
        // XOR of all is 0, we can consider all
        // subsets as one group.
        if (xs == 0)
            return (1 << (n - 1)) - 1;

        return 0;
    }

    // Driver program
    public static void Main()
    {
        int[] a = { 1, 2, 3 };
        int n = a.Length;
        Console.WriteLine(countgroup(a, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to count number
// of ways to split array into
// two groups such that each
// group has equal XOR value

// Return the count number of
```

```
// ways to split array into
// two groups such that each
// grouphas equal XOR value.
function countgroup($a, $n)
{
    $xs = 0;
    for ($i = 0; $i < $n; $i++)
        $xs = $xs ^ $a[$i];

    // We can split only if XOR is 0. Since
    // XOR of all is 0, we can consider all
    // subsets as one group.
    if ($xs == 0)
        return (1 << ($n - 1)) - 1;

    return 0;
}

// Driver Code
$a = array(1, 2, 3);
$n = count($a);
echo countgroup($a, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
3
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/ways-split-array-two-groups-xor-value/

Chapter 338

What are the differences between bitwise and logical AND operators in C/C++?

What are the differences between bitwise and logical AND operators in C/C++? - GeeksforGeeks

A Bitwise And operator is represented as '&' and a logical operator is represented as '&&'. Following are some basic differences between the two operators.

a) The logical and operator '&&' expects its operands to be boolean expressions (either 1 or 0) and returns a boolean value.
The bitwise and operator '&' works on Integral (short, int, unsigned, char, bool, unsigned char, long) values and return Integral value.

```
 int main()
{
    int x = 3;  //...0011
    int y = 7;  //...0111

    // A typical use of '&&'
    if (y > 1 && y > x)
      printf("y is greater than 1 AND y\n");

    // A typical use of '&'
    int z = x & y;   // 0011

    printf ("z = %d", z);

    return 0;
}
```

Output

```
y is greater than 1 AND y
z = 3.
```

b) If an integral value is used as an operand for '&&' which is supposed to work on boolean values, following rule is used in C.
.....A zero is considered as false and non-zero is considered as true.
For example in the following program x and y are considered as 1.

```
 // Example that uses non-boolean expression as
// operand for '&&'
int main()
{
   int x = 2, y = 5;
   printf("%d", x&&y);
   return 0;
}
```

Output

```
1
```

It is compiler error to use non-integral expression as operand for bitwise &. For example the following program shows compiler error.

```
 // Example that uses non-integral expression as
// operator for '&'
int main()
{
   float x = 2.0, y = 5.0;
   printf("%d", x&y);
   return 0;
}
```

Output:

```
error: invalid operands to binary & (have 'float' and 'float')
```

c) The '&&' operator doesn't evaluate second operand if first operand becomes false. Similarly '||' doesn't evaluate second operand when first operand becomes true. The bitwise '&' and '|' operators always evaluate their operands.

```
 int main()
{
   int x = 0;

   // 'Geeks in &&' is NOT printed because x is 0
   printf("%d\n", (x && printf("Geeks in && ")) );

   // 'Geeks in &' is  printed
   printf("%d\n", (x  & printf("Geeks in & ")) );

   return 0;
}
```

Output:

```
0
Geeks in & 0
```

The same differences are there between logical OR '||' and bitwise OR '|'.

This article is contributed by **Ujjwal Jain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/what-are-the-differences-between-bitwise-and-logical-and-operators-in-cc/

Chapter 339

Write a function that returns 2 for input 1 and returns 1 for 2

Write a function that returns 2 for input 1 and returns 1 for 2 - GeeksforGeeks

Write a function which returns 1 that 2 is passed and return 2 when 1 is passed.

Source: Adobe Interview Experience | Set 19 (For MTS)

A **simple solution** is to compare the passed value with 1 and 2.

```
 int invert(int x)
{
   if (x == 1) return 2;
   else return 1;
}
```

Another solution is to **use subtraction**

```
 int invertSub(int x)
{
   return (3-x);
}
```

We can also **use bitwise xor operator**.

```
 int invertXOR(int x)
{
   return (x ^ 1 ^ 2);
}
```

This article is contributed by **Anuj**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/write-function-returns-2-input-1-returns-1-2/

Chapter 340

Write an Efficient C Program to Reverse Bits of a Number

Write an Efficient C Program to Reverse Bits of a Number - GeeksforGeeks

Given an unsigned integer, reverse all bits of it and return the number with reversed bits.

```
Input : n = 1
Output : 2147483648
On a machine with size of unsigned
bit as 32. Reverse of 0....001 is
100....0.

Input : n = 2147483648
Output : 1
```

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(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));
    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. Please referReverse bits using lookup table in O(1) time for details.

Source :
https://graphics.stanford.edu/~seander/bithacks.html

Improved By : oyemohit

Source

https://www.geeksforgeeks.org/write-an-efficient-c-program-to-reverse-bits-of-a-number/

Chapter 341

Write an Efficient Method to Check if a Number is Multiple of 3

Write an Efficient Method to Check if a Number is Multiple of 3 - GeeksforGeeks

The very first solution that comes to our mind is the one that we learned in school. If sum of digits in a number is multiple of 3 then number is multiple of 3 e.g., for 612 sum of digits is 9 so it's a multiple of 3. But this solution is not efficient. You have to get all decimal digits one by one, add them and then check if sum is multiple of 3.

There is a pattern in binary representation of the number that can be used to find if number is a multiple of 3. If difference between count of odd set bits (Bits set at odd positions) and even set bits is multiple of 3 then is the number.

Example : 23 (00..10111)
1) Get count of all set bits at odd positions (For 23 it's 3).
2) Get count of all set bits at even positions (For 23 it's 1).
3) If difference of above two counts is a multiple of 3 then number is also a multiple of 3.

(For 23 it's 2 so 23 is not a multiple of 3)

Take some more examples like 21, 15, etc…

```
Algorithm: isMutlipleOf3(n)
1) Make n positive if n is negative.
2) If number is 0 then return 1
3) If number is 1 then return 0
4) Initialize: odd_count = 0, even_count = 0
5) Loop while n != 0
    a) If rightmost bit is set then increment odd count.
    b) Right-shift n by 1 bit
```

```
    c) If rightmost bit is set then increment even count.
    d) Right-shift n by 1 bit
6) return isMutlipleOf3(odd_count - even_count)
```

Proof:
Above can be proved by taking the example of 11 in decimal numbers. (In this context 11 in decimal numbers is same as 3 in binary numbers)
If difference between sum of odd digits and even digits is multiple of 11 then decimal number is multiple of 11. Let's see how.

Let's take the example of 2 digit numbers in decimal
AB = 11A -A + B = 11A + (B – A)
So if (B – A) is a multiple of 11 then is AB.

Let us take 3 digit numbers.

ABC = 99A + A + 11B – B + C = (99A + 11B) + (A + C – B)
So if (A + C – B) is a multiple of 11 then is (ABC)

Let us take 4 digit numbers now.
ABCD = 1001A + D + 11C – C + 999B + B – A
= (1001A – 999B + 11C) + (D + B – A -C)
So, if (B + D – A – C) is a multiple of 11 then is ABCD.

This can be continued for all decimal numbers.
Above concept can be proved for 3 in binary numbers in the same way.

Time Complexity: O(logn)

Program:

C++

```
 // CPP program to check if n is a multiple of 3
#include<bits/stdc++.h>
using namespace std;

/* Function to check if n is a multiple of 3*/
int isMultipleOf3(int n)
{
    int odd_count = 0;
    int even_count = 0;

    /* Make no positive if +n is multiple of 3
    then is -n. We are doing this to avoid
    stack overflow in recursion*/
    if(n < 0) n = -n;
    if(n == 0) return 1;
    if(n == 1) return 0;
```

```
    while(n)
    {
        /* If odd bit is set then
        increment odd counter */
        if(n & 1)
        odd_count++;
        n = n>>1;

        /* If even bit is set then
        increment even counter */
        if(n & 1)
            even_count++;
        n = n>>1;
    }

    return isMultipleOf3(abs(odd_count - even_count));
}

/* Program to test function isMultipleOf3 */
int main()
{
    int num = 24;
    if (isMultipleOf3(num))
        printf("%d is multiple of 3", num);
    else
        printf("%d is not a multiple of 3", num);
    return 0;
}
```

Java

```
 // Java program to check if
// n is a multiple of 3
import java.lang.*;
import java.util.*;

class GFG
{
 // Function to check if n
 // is a multiple of 3
 static int isMultipleOf3(int n)
  {
    int odd_count = 0;
    int even_count = 0;

    /* Make no positive if +n is multiple
    of 3 then is -n. We are doing this to
    avoid stack overflow in recursion*/
```

```
    if(n < 0) n = -n;
    if(n == 0) return 1;
    if(n == 1) return 0;

    while(n != 0)
    {
        /* If odd bit is set then
        increment odd counter */
        if((n & 1) != 0 )
        odd_count++;
        n = n>>1;

        /* If even bit is set then
        increment even counter */
        if((n & 1) != 0)
            even_count++;

        n = n>>1;
    }

    return isMultipleOf3
    (Math.abs(odd_count - even_count));
}

/* Program to test function isMultipleOf3 */
    public static void main(String[] args)
    {
    int num = 24;

    if (isMultipleOf3(num) != 0)
        System.out.println(num +
        " is multiple of 3");
    else
        System.out.println(num +
        " is not a multiple of 3");
}
}

// This code is contributed by Sahil_Bansall
```

Python3

```
 # Python profram to check if n is a multiple of 3

# Function to check if n is a multiple of 3
def isMultipleOf3(n):

    odd_count = 0
```

```
    even_count = 0

    # Make no positive if +n is multiple of 3
    # then is -n. We are doing this to avoid
    # stack overflow in recursion
    if(n < 0):
        n = -n
    if(n == 0):
        return 1
    if(n == 1):
        return 0

    while(n):

        # If odd bit is set then
        # increment odd counter
        if(n & 1):
            odd_count += 1
        n = n >> 1

        # If even bit is set then
        # increment even counter
        if(n & 1):
            even_count += 1
        n = n >> 1

    return isMultipleOf3(abs(odd_count - even_count))

# Program to test function isMultipleOf3
num = 24
if (isMultipleOf3(num)):
    print(num, 'is multiple of 3')
else:
    print(num, 'is not a multiple of 3')

# This code is contributed by Danish Raza
```

C#

```
 // C# program to check if
// n is a multiple of 3
using System;

class GFG {

    // Function to check if n
    // is a multiple of 3
    static int isMultipleOf3(int n)
```

```
    {
        int odd_count = 0, even_count = 0;

        /* Make no positive if +n is multiple
        of 3 then is -n. We are doing this to
        avoid stack overflow in recursion*/
        if (n < 0) n = -n;
        if (n == 0) return 1;
        if (n == 1) return 0;

        while (n != 0) {

            /* If odd bit is set then
            increment odd counter */
            if ((n & 1) != 0)
                odd_count++;

            n = n >> 1;

            /* If even bit is set then
            increment even counter */
            if ((n & 1) != 0)
                even_count++;

            n = n >> 1;
        }

        return isMultipleOf3(Math.Abs(odd_count - even_count));
    }

    // Driver Code
    public static void Main()
    {
        int num = 24;

        if (isMultipleOf3(num) != 0)
            Console.Write(num + " is multiple of 3");
        else
            Console.Write(num + " is not a multiple of 3");
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to check if n
```

```
// is a multiple of 3


// Function to check if n
// is a multiple of 3
function isMultipleOf3( $n)
{
    $odd_count = 0;
    $even_count = 0;

    // Make no positive if +n
    // is multiple of 3 then is -n.
    // We are doing this to avoid
    // stack overflow in recursion
    if($n < 0) $n = -$n;
    if($n == 0) return 1;
    if($n == 1) return 0;

    while($n)
    {
        // If odd bit is set then
        // increment odd counter
        if($n & 1)
        $odd_count++;
        $n = $n>>1;

        // If even bit is set then
        // increment even counter
        if($n & 1)
            $even_count++;
        $n = $n >> 1;
    }

    return isMultipleOf3(abs($odd_count -
                             $even_count));
}

// Driver Code
$num = 24;
if (isMultipleOf3($num))
    echo $num, "is multiple of 3";
else
    echo $num,"is not a multiple of 3";

// This code is contributed by vt_m.
?>
```

Output :

```
24 is multiple of 3
```

Related Articles:
Check divisibility in a binary stream
DFA based division

Improved By : Deepak Koli, vt_m

Source

https://www.geeksforgeeks.org/write-an-efficient-method-to-check-if-a-number-is-multiple-of-3/

Chapter 342

Write your own strcmp that ignores cases

Write your own strcmp that ignores cases - GeeksforGeeks

Write a modified strcmp function which ignores cases and returns -1 if s1 < s2, 0 if s1 = s2, else returns 1. For example, your strcmp should consider "GeeksforGeeks" and "geeksforgeeks" as same string.

Source: Microsoft Interview Set 5

Following solution assumes that characters are represented using ASCII representation, i.e., codes for 'a', 'b', 'c', ... 'z' are 97, 98, 99, ... 122 respectively. And codes for 'A', "B", 'C', ... 'Z' are 65, 66, ... 95 respectively.

Following are the detailed steps.
1) Iterate through every character of both strings and do following for each character.
...**a)** If str1[i] is same as str2[i], then continue.
...**b)** If inverting the 6th least significant bit of str1[i] makes it same as str2[i], then continue. For example, if str1[i] is 65, then inverting the 6th bit will make it 97. And if str1[i] is 97, then inverting the 6th bit will make it 65.
...**c)** If any of the above two conditions is not true, then break.
2) Compare the last (or first mismatching in case of not same) characters.

```
 #include <stdio.h>

/* implementation of strcmp that ignores cases */
int ic_strcmp(char *s1, char *s2)
{
    int i;
    for (i = 0; s1[i] && s2[i]; ++i)
    {
        /* If characters are same or inverting the
           6th bit makes them same */
```

```
        if (s1[i] == s2[i] || (s1[i] ^ 32) == s2[i])
           continue;
        else
           break;
    }

    /* Compare the last (or first mismatching in
       case of not same) characters */
    if (s1[i] == s2[i])
        return 0;

    // Set the 6th bit in both, then compare
    if ((s1[i] | 32) < (s2[i] | 32))
        return -1;
    return 1;
}

// Driver program to test above function
int main(void)
{
    printf("ret: %d\n", ic_strcmp("Geeks", "apple"));
    printf("ret: %d\n", ic_strcmp("", "ABCD"));
    printf("ret: %d\n", ic_strcmp("ABCD", "z"));
    printf("ret: %d\n", ic_strcmp("ABCD", "abcdEghe"));
    printf("ret: %d\n", ic_strcmp("GeeksForGeeks", "gEEksFORGeEKs"));
    printf("ret: %d\n", ic_strcmp("GeeksForGeeks", "geeksForGeeks"));
    return 0;
}
```

Output:

```
ret: 1
ret: -1
ret: -1
ret: -1
ret: 0
ret: 0
```

This article is compiled by **Narendra Kangralkar**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/write-your-own-strcmp-which-ignores-cases/

Chapter 343

XNOR of two numbers

XNOR of two numbers - GeeksforGeeks

XNOR gives the reverse of XOR if binary bit.

```
First bit | Second bit | XNOR
0             0           1
0             1           0
1             0           0
1             1           1
It gives 1 if bits are same else if
bits are different it gives 0.
```

It is reverse of XOR but we can't implement it directly so this is the program for implement XNOR

Examples:

```
Input  : 10 20
Output : 1
Binary of 20 is 10100
Binary of 10 is  1010
So the XNOR is  00001
So output is 1

Input  : 10 10
Output : 15
Binary of 10 is  1010
Binary of 10 is  1010
So the XNOR is   1111
So output is 15
```

First Method:- (O(logn)) In this solution we check one bit at a time. If two bits are same, we put 1 in result, else we put 0.

Let's understand it with below code

C++

```
 // CPP program to find
// XNOR of two numbers
#include <iostream>
using namespace std;

// log(n) solution
int xnor(int a, int b)
{
    // Make sure a is larger
    if (a < b)
        swap(a, b);

    if (a == 0 && b == 0)
        return 1;

    int a_rem = 0; // for last bit of a
    int b_rem = 0; // for last bit of b

    // counter for count bit
    // and set bit  in xnornum
    int count = 0;

    // to make new xnor number
    int xnornum = 0;

    // for set bits in new xnor number
    while (a)
    {
        // get last bit of a
        a_rem = a & 1;

        // get last bit of b
        b_rem = b & 1;

        // Check if current two
        // bits are same
        if (a_rem == b_rem)
            xnornum |= (1 << count);

        // counter for count bit
        count++;
        a = a >> 1;
```

```
        b = b >> 1;
    }
    return xnornum;
}

// Driver code
int main()
{
    int a = 10, b = 50;
    cout << xnor(a, b);
    return 0;
}
```

Java

```
 // Java program to find
// XNOR of two numbers
import java.util.*;
import java.lang.*;

public class GfG {

    public static int xnor(int a, int b)
    {
        // Make sure a is larger
        if (a < b) {
            // swapping a and b;
            int t = a;
            a = b;
            b = t;
        }

        if (a == 0 && b == 0)
            return 1;

        // for last bit of a
        int a_rem = 0;

        // for last bit of b
        int b_rem = 0;

        // counter for count bit
        // and set bit in xnornum
        int count = 0;

        // to make new xnor number
        int xnornum = 0;
```

```
        // for set bits in new xnor number
        while (true) {
            // get last bit of a
            a_rem = a & 1;

            // get last bit of b
            b_rem = b & 1;

            // Check if current two bits are same
            if (a_rem == b_rem)
                xnornum |= (1 << count);

            // counter for count bit
            count++;
            a = a >> 1;
            b = b >> 1;
            if (a < 1)
                break;
        }
        return xnornum;
    }

    // Driver function
    public static void main(String argc[])
    {
        int a = 10, b = 50;
        System.out.println(xnor(a, b));
    }
}
```

Python3

```
 # Python3 program to find XNOR
# of two numbers
import math

def swap(a,b):

    temp=a
    a=b
    b=temp

# log(n) solution
def xnor(a, b):

    # Make sure a is larger
    if (a < b):
        swap(a, b)
```

```
    if (a == 0 and b == 0) :
        return 1;

    # for last bit of a
    a_rem = 0

    # for last bit of b
    b_rem = 0

    # counter for count bit and
    #  set bit in xnor num
    count = 0

    # for make new xnor number
    xnornum = 0

    # for set bits in new xnor
    # number
    while (a!=0) :

        # get last bit of a
        a_rem = a & 1

        # get last bit of b
        b_rem = b & 1

        # Check if current two
        # bits are same
        if (a_rem == b_rem):
            xnornum |= (1 << count)

        # counter for count bit
        count=count+1

        a = a >> 1
        b = b >> 1

    return xnornum;

# Driver method
a = 10
b = 50
print(xnor(a, b))

# This code is contributed by Gitanjali
```

C#

```
 // C# program to find
// XNOR of two numbers
using System;

public class GfG {

    public static int xnor(int a, int b)
    {
        // Make sure a is larger
        if (a < b) {
            // swapping a and b;
            int t = a;
            a = b;
            b = t;
        }

        if (a == 0 && b == 0)
            return 1;

        // for last bit of a
        int a_rem = 0;

        // for last bit of b
        int b_rem = 0;

        // counter for count bit
        // and set bit in xnornum
        int count = 0;

        // to make new xnor number
        int xnornum = 0;

        // for set bits in new xnor number
        while (true) {
            // get last bit of a
            a_rem = a & 1;

            // get last bit of b
            b_rem = b & 1;

            // Check if current two bits are same
            if (a_rem == b_rem)
                xnornum |= (1 << count);

            // counter for count bit
            count++;
            a = a >> 1;
            b = b >> 1;
```

```
            if (a < 1)
                break;
        }
        return xnornum;
    }

    // Driver function
    public static void Main()
    {
        int a = 10, b = 50;
        Console.WriteLine(xnor(a, b));
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP program to find
// XNOR of two numbers

// log(n) solution
function xnor($a, $b)
{

    // Make sure a is larger
    if ($a < $b)
        list($a, $b) = array($b, $a);

    if ($a == 0 && $b == 0)
        return 1;

    // for last bit of a
    $a_rem = 0;

    // for last bit of b
    $b_rem = 0;

    // counter for count bit
    // and set bit in xnornum
    $count = 0;

    // to make new xnor number
    $xnornum = 0;

    // for set bits in
    // new xnor number
```

```
    while ($a)
    {

        // get last bit of a
        $a_rem = $a & 1;

        // get last bit of b
        $b_rem = $b & 1;

        // Check if current two
        // bits are same
        if ($a_rem == $b_rem)
            $xnornum |= (1 << $count);

        // counter for count bit
        $count++;
        $a = $a >> 1;
        $b = $b >> 1;
    }
    return $xnornum;
}

    // Driver code
    $a = 10;
    $b = 50;
    echo xnor($a, $b);

// This code is contributed by mits.
?>
```

Output:

```
7
```

Second Method:- O(1)
1) Find maximum of two given numbers.
2) Toggle all bits in higher of two numbers.
3) Return XOR of original smaller number and modified larger number.

C++

```
 // CPP program to find XNOR of two numbers.
#include <iostream>
using namespace std;
```

```
// Please refer below post for details of this
// function
// https://www.geeksforgeeks.org/toggle-bits-significant-bit/
int togglebit(int n)
{
    if (n == 0)
        return 1;

    // Make a copy of n as we are
    // going to change it.
    int i = n;

    // Below steps set bits after
    // MSB (including MSB)

    // Suppose n is 273 (binary
    // is 100010001). It does following
    // 100010001 | 010001000 = 110011001
    n |= n >> 1;

    // This makes sure 4 bits
    // (From MSB and including MSB)
    // are set. It does following
    // 110011001 | 001100110 = 111111111
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    return i ^ n;
}

// Returns XNOR of num1 and num2
int XNOR(int num1, int num2)
{
    // if num2 is greater then
    // we swap this number in num1
    if (num1 < num2)
        swap(num1, num2);
    num1 = togglebit(num1);

    return num1 ^ num2;
}

// Driver code
int main()
{
    int num1 = 10, num2 = 20;
```

```
    cout << XNOR(num1, num2);
    return 0;
}
```

Java

```
 // Java program to find XNOR
// of two numbers
import java.io.*;

class GFG {

    // Please refer below post for
    // details of this function
    // https://www.geeksforgeeks.org/toggle-bits-significant-bit/
    static int togglebit(int n)
    {
        if (n == 0)
            return 1;

        // Make a copy of n as we are
        // going to change it.
        int i = n;

        // Below steps set bits after
        // MSB (including MSB)

        // Suppose n is 273 (binary
        // is 100010001). It does following
        // 100010001 | 010001000 = 110011001
        n |= n >> 1;

        // This makes sure 4 bits
        // (From MSB and including MSB)
        // are set. It does following
        // 110011001 | 001100110 = 111111111
        n |= n >> 2;
        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;

        return i ^ n;
    }

    // Returns XNOR of num1 and num2
    static int xnor(int num1, int num2)
    {
        // if num2 is greater then
```

```
        // we swap this number in num1
        if (num1 < num2)
        {
            int temp = num1;
            num1 = num2;
            num2 = temp;
        }

        num1 = togglebit(num1);

        return num1 ^ num2;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int a = 10, b = 20;
        System.out.println(xnor(a, b));
    }
}

// This code is contributed by Gitanjali
```

Python

```
 # python program to find XNOR of two numbers
import math

# Please refer below post for details of this function
# https://www.geeksforgeeks.org/toggle-bits-significant-bit/
def togglebit( n):

    if (n == 0):
        return 1

    # Make a copy of n as we are
    # going to change it.
    i = n

    # Below steps set bits after
    # MSB (including MSB)

    # Suppose n is 273 (binary
    # is 100010001). It does following
    # 100010001 | 010001000 = 110011001
    n = n|(n >> 1)

    # This makes sure 4 bits
```

```
    # (From MSB and including MSB)
    # are set. It does following
    # 110011001 | 001100110 = 111111111
    n |= n >> 2
    n |= n >> 4
    n |= n >> 8
    n |= n >> 16

    return i ^ n

# Returns XNOR of num1 and num2
def xnor( num1, num2):

    # Make sure num1 is larger
    if (num1 < num2):
        temp = num1
        num1 = num2
        num2 = temp
    num1 = togglebit(num1)

    return num1 ^ num2

# Driver code
a = 10
b = 20
print (xnor(a, b))

# This code is contributed by 'Gitanjali'.
```

C#

```
 // C# program to find XNOR
// of two numbers
using System;

class GFG {

    // Please refer below post for
    // details of this function
    // https://www.geeksforgeeks.org/toggle-bits-significant-bit/
    static int togglebit(int n)
    {
        if (n == 0)
            return 1;

        // Make a copy of n as we are
        // going to change it.
        int i = n;
```

```
        // Below steps set bits after
        // MSB (including MSB)

        // Suppose n is 273 (binary
        // is 100010001). It does following
        // 100010001 | 010001000 = 110011001
        n |= n >> 1;

        // This makes sure 4 bits
        // (From MSB and including MSB)
        // are set. It does following
        // 110011001 | 001100110 = 111111111
        n |= n >> 2;
        n |= n >> 4;
        n |= n >> 8;
        n |= n >> 16;

        return i ^ n;
    }

    // Returns XNOR of num1 and num2
    static int xnor(int num1, int num2)
    {
        // if num2 is greater then
        // we swap this number in num1
        if (num1 < num2)
        {
            int temp = num1;
            num1 = num2;
            num2 = temp;
        }

        num1 = togglebit(num1);

        return num1 ^ num2;
    }

    // Driver program
    public static void Main()
    {
        int a = 10, b = 20;
        Console.WriteLine(xnor(a, b));
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP program to find XNOR
// of two numbers.

// Please refer below post
// for details of this function
// https://www.geeksforgeeks.org/toggle-bits-significant-bit/

function togglebit($n)
{
    if ($n == 0)
        return 1;

    // Make a copy of n as we are
    // going to change it.
    $i = $n;

    // Below steps set bits after
    // MSB (including MSB)

    // Suppose n is 273 (binary
    // is 100010001). It does following
    // 100010001 | 010001000 = 110011001
    $n |= $n >> 1;

    // This makes sure 4 bits
    // (From MSB and including MSB)
    // are set. It does following
    // 110011001 | 001100110 = 111111111
    $n |= $n >> 2;
    $n |= $n >> 4;
    $n |= $n >> 8;
    $n |= $n >> 16;

    return $i ^ $n;
}

// Returns XNOR of num1 and num2
function XNOR($num1, $num2)
{

    // if num2 is greater then
    // we swap this number in num1
    if ($num1 < $num2)
        list($num1, $num2)=array($num2, $num1);
    $num1 = togglebit($num1);
```

```
    return $num1 ^ $num2;
}

// Driver code
$num1 = 10;
$num2 = 20;
echo XNOR($num1, $num2);

// This code is contributed by Smitha.
?>
```

Output :

```
1
```

Improved By : Mithun Kumar, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/xnor-two-numbers/

Chapter 344

XOR counts of 0s and 1s in binary representation

XOR counts of 0s and 1s in binary representation - GeeksforGeeks

Given a number, the task is to find XOR of count of 0s and count of 1s in binary representation of a given number.
Examples:

```
Input  : 5
Output : 3
Binary representation : 101
Count of 0s = 1,
Count of 1s = 2
1 XOR 2 = 3.

Input  : 7
Output : 3
Binary representation : 111
Count of 0s = 0
Count of 1s = 3
0 XOR 3 = 3.
```

The idea is simple, we traverse through all bits of a number, count 0s and 1s and finally return XOR of two counts.

C++

```
 // C++ program to find XOR of counts 0s and 1s in
// binary representation of n.
#include<iostream>
```

```
using namespace std;

// Returns XOR of counts 0s and 1s in
// binary representation of n.
int countXOR(int n)
{
    int count0 = 0, count1 = 0;
    while (n)
    {
        //calculating count of zeros and ones
        (n % 2 == 0) ? count0++ :count1++;
        n /= 2;
    }
    return (count0 ^ count1);
}

// Driver Program
int main()
{
    int n = 31;
    cout << countXOR (n);
    return 0;
}
```

Java

```
 // Java program to find XOR of counts 0s
// and 1s in binary representation of n.

class GFG {

    // Returns XOR of counts 0s and 1s
    // in binary representation of n.
    static int countXOR(int n)
    {
        int count0 = 0, count1 = 0;
        while (n != 0)
        {
            //calculating count of zeros and ones
            if(n % 2 == 0)
            count0++ ;
            else
            count1++;
            n /= 2;
        }
        return (count0 ^ count1);
    }
```

```
    // Driver Program
    public static void main(String[] args)
    {
        int n = 31;
        System.out.println(countXOR (n));
    }
}

// This code is contributed by prerna saini
```

Python3

```
 # Python3 program to find XOR of counts 0s
# and 1s in binary representation of n.

# Returns XOR of counts 0s and 1s
# in binary representation of n.
def countXOR(n):

    count0, count1 = 0, 0
    while (n != 0):

        # calculating count of zeros and ones
        if(n % 2 == 0):
            count0 += 1
        else:
            count1 += 1
        n //= 2

    return (count0 ^ count1)

# Driver Code
n = 31
print(countXOR(n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to find XOR of counts 0s
// and 1s in binary representation of n.
using System;

class GFG {

    // Returns XOR of counts 0s and 1s
    // in binary representation of n.
```

```
    static int countXOR(int n)
    {
        int count0 = 0, count1 = 0;
        while (n != 0)
        {

            // calculating count of zeros
            // and ones
            if(n % 2 == 0)
                count0++ ;
            else
                count1++;

            n /= 2;
        }

        return (count0 ^ count1);
    }

    // Driver Program
    public static void Main()
    {

        int n = 31;

        Console.WriteLine(countXOR (n));
    }
}

// This code is contributed by Anant Agarwal.
```

Output:

```
 5
```

One observation is, for a number of the form **2^x − 1**, the output is always x. We can directly produce answer for this case by first checking n+1 is apower of two or not.

Source

https://www.geeksforgeeks.org/xor-counts-0s-1s-binary-representation/

Chapter 345

XOR Encryption by Shifting Plaintext

XOR Encryption by Shifting Plaintext - GeeksforGeeks

Here is a cipher algorithm, based on hexadecimal strings that is implemented by **XORing** the given plaintext, **N** number of times where **N** is its length. But, the catch is that every next XOR operation is done after shifting the consecutive plain text **entry to the right**. A sample operation is shown below :

a	b	c	d			
	a	b	c	d		
		a	b	c	d	
			a	b	c	d
a	1	d	0	a	1	d

Suppose the password is **'abcd'** then the hexadecimal text is calculated as **a1d0a1d** by XORing the password with itself N times i.e. 4 times in this case.

Similarly if the password is **'636f646572'**, then

6	3	6	f	6	4	6	5	7	2								
	6	3	6	f	6	4	6	5	7	2							
		6	3	6	f	6	4	6	5	7	2						
			6	3	6	f	6	4	6	5	7	2					
				6	3	6	f	6	4	6	5	7	2				
					6	3	6	f	6	4	6	5	7	2			
						6	3	6	f	6	4	6	5	7	2		
							6	3	6	f	6	4	6	5	7	2	
								6	3	6	f	6	4	6	5	7	2
6	5	3	c	a	e	8	d	a	8	e	d	b	4	2	6	0	5	2

653cae8da8edb426052 is the hexadecimal text.

So, the problem statement is to create a decryption algorithm (in any programming language) and deduce the plain text from the given hexadecimal string.

Examples :

```
Input : a1d0a1d
Output : abcd
abcd once coded will return a1d0a1d

Input : 653cae8da8edb426052
Output : 636f646572
```

Approach : The key ingredient in encrypting and decrypting is in the properties of **XOR**. XOR is a bitwise operation where the result is **0** if the two possible inputs are **same** but **1** when the inputs are **different**. The XOR table is given below for reference :

Inputs

Outputs

X

Y

Z

0

0

0

0

1

1

1

0

1

1

1

0

An important and useful property of XOR that is widely popular in cryptography is that in case of multiple XORing of numbers (say **M** numbers), if we know only the **M − 1** numbers (one is unknown) along with the **XOR result** then, we can easily calculate the missing number by XORing the known numbers and the XOR result. This property is discussed with the following hexadecimal numbers :

Operation	Result
D ^ 2	F
D ^ 2 ^ F	0
D ^ 2 ^ F ^ 5	5
D ^ 2 ^ F ^ 5 ^ 3	6
Assuming 3 to be the unknown element and XORing the result = 9	
D ^ 2 ^ F ^ 5 ^ 6	3

We shall be using the above listed property the most in course of this problem. Now, if we look at the encryption diagram of **'abcd'** at the base it is just the repeated XORing of the digits. The rightmost digit is **d** and the rightmost digit of the **'abcd'** is **d** as well so the last digit of both plaintext and hexstring is the **same**. The next digit is **1** which is calculated by XORing the second right digit of **abcd** and the previous digit i.e. $\mathbf{1 = d \hat{} c}$ using the property we know the plain text digit can be deduced as $\mathbf{d \hat{} 1 = c}$. Similarly, the next digit is **a** which is found by $\mathbf{d \hat{} c \hat{} b = a}$. We only need to do this till the half of the hex string as the rest is **symmetrical so they are not required**.

Position	Operation	Hex Character	Deduction	Plain text
7	d	d	Last digit is same for both	d
6	d ^ c	1	d ^ 1 = c	c
5	d ^ c ^ b	a	d ^ c ^ a = b	b
4	d ^ c ^ b ^ a	0	d ^ c ^ a ^ 0 = a	a
The operation on rest is not required as they are symmetrical				

Hex string: a1d0a1d Plain text: **abcd**

Below is the implementation of above approach :

```
# Implementation in Python 3
```

```
# Hex String variable
hex_s = '653cae8da8edb426052'

# Plain text variable
plain = ''

# variable to store the XOR
# of previous digits
x = 0

l = len(hex_s)

# Loop for loop from the end to
# the mid section of the string
for i in range(l - 1, int(l / 2) - 1, -1):

    # calculation of the plaintext digit
    y = x^int(hex_s[i], 16)

    # calculation of XOR chain
    x = x^y
    plain = hex(y)[-1] + plain

print(plain)
```

Output:

```
636f646572
```

Source

https://www.geeksforgeeks.org/xor-encryption-shifting-plaintext/

Chapter 346

XOR of Sum of every possible pair of an array

XOR of Sum of every possible pair of an array - GeeksforGeeks

Given an array A of size n. the task is to generate a new sequence B with size N^2 having elements sum of every pair of array A and find the xor value of sum of all the pairs formed. Note: Here (A[i], A[i]), (A[i], A[j]), (A[j], A[i]) all are considered as different pairs.

Examples:

```
Input: arr = {1, 5, 6}
Output: 4
B[2*2] = { 1+1, 1+5, 1+6, 5+1, 5+5, 5+6, 6+1, 6+5, 6+6}
B[4] = { 2, 6, 7, 6, 10, 11, 7, 11, 12}
So, 2 ^ 6 ^ 7 ^ 6 ^ 10 ^ 11 ^ 7 ^ 6 ^ 11 ^ 12 = 4

Input :1, 2
Output :6
```

A **Naive** approach is to run two loops. Consider each and every pair, take their sum and calculate the xor value of the sum of all the pairs.

An **Efficient** approach is based upon the fact that xor of same values is 0.
All the pairs like (a[i], a[j]) and (a[j], a[i]) will have same sum. So, their xor values will be 0. Only the pairs like (a[i], a[i]) will give the different result. So, take the xor of all the elements of given array and multiply it by 2.

C++

```
 // C++ program to find XOR of
// sum of every possible pairs
// in an array
```

```
#include <bits/stdc++.h>
using namespace std;

// Function to find XOR of sum
// of all pairs
int findXor(int arr[], int n)
{

    // Calculate xor of all the elements
    int xoR = 0;
    for (int i = 0; i < n; i++) {
        xoR = xoR ^ arr[i];
    }

    // Return twice of xor value
    return xoR * 2;
}

// Drivers code
int main()
{
    int arr[3] = { 1, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << findXor(arr, n);

    return 0;
}
```

Java

```
 // Java program to find XOR of
// sum of every possible pairs
// in an array
import java.io.*;

class GFG {

    // Function to find XOR of sum
    // of all pairs
    static int findXor(int arr[], int n)
    {

        // Calculate xor of all the
        // elements
        int xoR = 0;
        for (int i = 0; i < n; i++) {
            xoR = xoR ^ arr[i];
```

```
        }

        // Return twice of xor value
        return xoR * 2;
    }

    // Drivers code
    public static void main (String[] args)
    {
        int arr[] = { 1, 5, 6 };
        int n = arr.length;
        System.out.println( findXor(arr, n));
    }
}

// This code is contributed by anuj_67.
```

Python3

```
 # Python3 program to find
# XOR of sum of every
# possible pairs in an array

# Function to find XOR
# of sum of all pairs
def findXor(arr,n):

    # Calculate xor of
    # all the elements
    xoR = 0;
    for i in range (0, n ) :
        xoR = xoR ^ arr[i]

    # Return twice of
    # xor value
    return xoR * 2

# Driver code
arr = [ 1, 5, 6 ]
n = len(arr)
print(findXor(arr, n))

# This code is contributed
# by ihritik
```

C#

```
 // C# program to find XOR of
// sum of every possible pairs
// in an array
using System;

class GFG {

    // Function to find XOR of sum
    // of all pairs
    static int findXor(int []arr, int n)
    {

        // Calculate xor of all the
        // elements
        int xoR = 0;
        for (int i = 0; i < n; i++) {
            xoR = xoR ^ arr[i];
        }

        // Return twice of xor value
        return xoR * 2;
    }

    // Drivers code
    public static void Main ()
    {
        int []arr = { 1, 5, 6 };
        int n = arr.Length;
        Console.WriteLine( findXor(arr, n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to find XOR
// of sum of every possible
// pairs in an array

// Function to find XOR
// of sum of all pairs
function findXor($arr, $n)
{

    // Calculate xor of
    // all the elements
```

```
    $xoR = 0;
    for ($i = 0; $i < $n; $i++)
    {
        $xoR = $xoR ^ $arr[$i];
    }

    // Return twice
    // of xor value
    return $xoR * 2;
}

// Driver code
$arr = array(1, 5, 6);
$n = count($arr);

echo findXor($arr, $n);

// This code is contributed
// by anuj_67.
?>
```

Output:

```
4
```

Improved By : vt_m, ihritik

Source

https://www.geeksforgeeks.org/xor-sum-every-possible-pair-array/

Chapter 347

XOR of all subarray XORs | Set 2

XOR of all subarray XORs | Set 2 - GeeksforGeeks

Given an array of integers, we need to get total XOR of all subarray XORs where subarray XOR can be obtained by XORing all elements of it.

Examples :

```
Input : arr[] = [3, 5, 2, 4, 6]
Output : 7
Total XOR of all subarray XORs is,
(3) ^ (5) ^ (2) ^ (4) ^ (6)
(3^5) ^ (5^2) ^ (2^4) ^ (4^6)
(3^5^2) ^ (5^2^4) ^ (2^4^6)
(3^5^2^4) ^ (5^2^4^6) ^
(3^5^2^4^6) = 7

Input : arr[] = {1, 2, 3, 4}
Output : 0
Total XOR of all subarray XORs is,
(1) ^ (2) ^ (3) ^ (4) ^
(1^2) ^ (2^3) ^ (3^4) ^
(1^2^3) ^ (2^3^4) ^
(1^2^3^4) = 0
```

We have discussed a O(n) solution in below post.

XOR of all subarray XORs | Set 1

As discussed in above post, frequency of element at i-th index is given by (i+1)*(N-i), where N is the size of the array

There are 4 cases possible:

Case 1: i is odd, N is odd
Let i = 2k+1, N = 2m+1
freq[i] = ((2k+1)+1)*((2m+1)-(2k+1)) = 4(m-k)(k+1) = even

Case 2: i is odd, N is even
Let i = 2k+1, N = 2m
freq[i] = ((2k+1)+1)*((2m)-(2k+1)) = 2(k+1)(2m-2k-1) = even

Case 3: i is even, N is odd
Let i = 2k, N = 2m+1
freq[i] = ((2k)+1)*((2m+1)-(2k)) = 2k(2m-2k+1)+(2m-2k)+1 = odd

Case 4: i is even, N is even
Let i = 2k, N = 2m
freq[i] = ((2k)+1)*((2m)-(2k)) = 2(m-k)(2k+1) = even

From this, we can conclude that if total no.of elements in the array is even, then frequency of element at any position is even. So total XOR will be 0. And if total no. of elements are odd, then frequency of elements at even positions are odd and add positions are even. So we need to find only the XOR of elements at even positions.

Below is implementation of above idea :

C++

```
 // C++ program to get total xor of all subarray xors
#include <bits/stdc++.h>
using namespace std;

// Returns XOR of all subarray xors
int getTotalXorOfSubarrayXors(int arr[], int N)
{
    // if even number of terms are there, all
    // numbers will appear even number of times.
    // So result is 0.
    if (N % 2 == 0)
       return 0;

    // else initialize result by 0 as (a xor 0 = a)
    int res = 0;
    for (int i = 0; i<N; i+=2)
        res ^= arr[i];

    return res;
}

// Driver code to test above methods
int main()
```

```
{
    int arr[] = {3, 5, 2, 4, 6};
    int N = sizeof(arr) / sizeof(arr[0]);
    cout << getTotalXorOfSubarrayXors(arr, N);
    return 0;
}
```

Java

```
 // Java program to get total
// xor of all subarray xors
import java.io.*;

class GFG
{
    // Returns XOR of all
    // subarray xors
    static int getTotalXorOfSubarrayXors(int arr[],
                                         int N)
    {

    // if even number of terms are
    // there, all numbers will appear
    // even number of times. So result is 0.
    if (N % 2 == 0)
    return 0;

    // else initialize result
    // by 0 as (a xor 0 = a)
    int res = 0;
    for (int i = 0; i < N; i += 2)
        res ^= arr[i];

    return res;
    }

    // Driver Code
    public static void main (String[] args)
    {
    int arr[] = {3, 5, 2, 4, 6};
    int N = arr.length;

    System.out.println(
            getTotalXorOfSubarrayXors(arr, N));
    }
}

// This code is contributed by ajit
```

C#

```
 // C# program to get total
// xor of all subarray xors
using System;

class GFG
{

    // Returns XOR of all
    // subarray xors
    static int getTotalXorOfSubarrayXors(int []arr,
                                         int N)
    {

    // if even number of terms
    // are there, all numbers
    // will appear even number
    // of times. So result is 0.
    if (N % 2 == 0)
    return 0;

    // else initialize result
    // by 0 as (a xor 0 = a)
    int res = 0;
    for (int i = 0; i < N; i += 2)
        res ^= arr[i];

    return res;
    }

    // Driver Code
    static void Main()
    {
    int []arr = {3, 5, 2, 4, 6};
    int N = arr.Length;
    Console.Write(getTotalXorOfSubarrayXors(arr, N));
}
}

// This code is contributed by aj_36
```

PHP

```
 <?php
// PHP program to get total
// xor of all subarray xors
```

```
// Returns XOR of all subarray xors
function getTotalXorOfSubarrayXors($arr, $N)
{

    // if even number of terms
    // are there, all numbers
    // will appear even number
    // of times. So result is 0.
    if ($N % 2 == 0)
    return 0;

    // else initialize result
    // by 0 as (a xor 0 = a)
    $res = 0;
    for ($i = 0; $i < $N; $i += 2)
        $res ^= $arr[$i];

    return $res;
}

    // Driver Code
    $arr = array(3, 5, 2, 4, 6);
    $N = count($arr);
    echo getTotalXorOfSubarrayXors($arr, $N);

// This code is contributed by anuj_67.
?>
```

Output:

```
7
```

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/xor-subarray-xors-set-2/

Chapter 348

XOR of two numbers after making length of their binary representations equal

XOR of two numbers after making length of their binary representations equal - GeeksforGeeks

Given two numbers say a and b. Print their XOR after making the lengths of their binary representation equal by adding trailing zeros to the binary representation of smaller one.

Examples :

```
Input : a = 13, b = 5
Output : 7
Explanation : Binary representation of 13 is 1101 and
of 5 is 101. As the length of "101" is smaller,
so add a '0' to it making it "1010', to make
the length of binary representations equal.
XOR of 1010 and 1101 gives 0111 which is 7.

Input : a = 7, b = 5
Output : 2
Explanation : Since the length of binary representations
of 7 i.e, 111 and 5 i.e, 101 are same, hence simply
print XOR of a and b.
```

Approach : Count the number of bits in binary representation of smaller number out of a and b. If the number of bits in smaller number(say a) exceeds to that of larger number(say b), then apply left shift to the smaller number by the number of exceeding bits, i.e, a = a<<(exceeding bits). After applying left shift, trailing zeroes will be added at the end of

binary representation of smaller number to make the number of bits in binary representation of both the numbers equal. XOR both the binary representations to get the final result.

Below is the implementation of above method :

C++

```
 // C++ implementation to return
// XOR of two numbers after making
// length of their binary representation same
#include <bits/stdc++.h>
using namespace std;

// function to count the number
// of bits in binary representation
// of an integer
int count(int n)
{
    // initialize count
    int c = 0;

    // count till n is non zero
    while (n)
    {
        c++;

        // right shift by 1
        // i.e, divide by 2
        n = n>>1;
    }
    return c;
}

// function to calculate the xor of
// two numbers by adding trailing
// zeros to the number having less number
// of bits in its binary representation.
int XOR(int a, int b)
{
    // stores the minimum and maximum
    int c = min(a,b);
    int d = max(a,b);

    // left shift if the number of bits
    // are less in binary representation
    if (count(c) < count(d))
       c = c << ( count(d) - count(c) );
```

```
    return (c^d);
}

// driver code to check the above function
int main()
{
    int a = 13, b = 5;
    cout << XOR(a,b);
    return 0;
}
```

Java

```
 // Java implementation to return
// XOR of two numbers after making
// length of their binary representation same
import java.io.*;

class GFG {

    // function to count the number
    // of bits in binary representation
    // of an integer
    static int count(int n)
    {
        // initialize count
        int c = 0;

        // count till n is non zero
        while (n != 0)
        {
            c++;

            // right shift by 1
            // i.e, divide by 2
            n = n >> 1;
        }
        return c;
    }

    // function to calculate the xor of
    // two numbers by adding trailing
    // zeros to the number having less number
    // of bits in its binary representation.
    static int XOR(int a, int b)
    {
        // stores the minimum and maximum
        int c = Math.min(a, b);
```

```
        int d = Math.max(a, b);

        // left shift if the number of bits
        // are less in binary representation
        if (count(c) < count(d))
        c = c << ( count(d) - count(c) );

        return (c ^ d);
    }

    // driver code to check the above function
    public static void main(String args[])
    {
        int a = 13, b = 5;
        System.out.println(XOR(a, b));
    }
}

// This code is contributed by Nikita Tiwari.
```

Python3

```
 # Python 3 implementation to return XOR
# of two numbers after making length
# of their binary representation same

# Function to count the number of bits
# in binary representation of an integer
def count(n) :

    # initialize count
    c = 0

    # count till n is non zero
    while (n != 0) :
        c += 1

        # right shift by 1
        # i.e, divide by 2
        n = n >> 1

    return c

# Function to calculate the xor of
# two numbers by adding trailing
# zeros to the number having less number
# of bits in its binary representation.
def XOR(a, b) :
```

```
    # stores the minimum and maximum
    c = min(a, b)
    d = max(a, b)

    # left shift if the number of bits
    # are less in binary representation
    if (count(c) < count(d)) :
        c = c << ( count(d) - count(c) )

    return (c^d)

# Driver Code
a = 13; b = 5
print(XOR(a, b))


# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# implementation to return XOR of two
// numbers after making length of their
// binary representation same
using System;

class GFG {

    // function to count the number
    // of bits in binary representation
    // of an integer
    static int count(int n)
    {

        // initialize count
        int c = 0;

        // count till n is non zero
        while (n != 0)
        {

            c++;

            // right shift by 1
            // i.e, divide by 2
            n = n >> 1;
        }
```

```
        return c;
    }

    // function to calculate the xor of
    // two numbers by adding trailing
    // zeros to the number having less number
    // of bits in its binary representation.
    static int XOR(int a, int b)
    {

        // stores the minimum and maximum
        int c = Math.Min(a, b);
        int d = Math.Max(a, b);

        // left shift if the number of bits
        // are less in binary representation
        if (count(c) < count(d))
        c = c << ( count(d) - count(c) );

        return (c ^ d);
    }

    // driver code to check the above function
    public static void Main()
    {
        int a = 13, b = 5;

        Console.WriteLine(XOR(a, b));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// php implementation to return XOR
// of two numbers after making
// length of their binary
// representation same

// function to count the number
// of bits in binary representation
// of an integer
function count1($n)
{

    // initialize count
```

```
    $c = 0;

    // count till n is
    // non zero
    while ($n)
    {
        $c++;

        // right shift by 1
        // i.e, divide by 2
        $n = $n>>1;
    }
    return $c;
}

// function to calculate the xor of
// two numbers by adding trailing
// zeros to the number having less number
// of bits in its binary representation.
function XOR1($a, $b)
{

    // stores the minimum
    // and maximum
    $c = min($a,$b);
    $d = max($a,$b);

    // left shift if the number of bits
    // are less in binary representation
    if (count1($c) < count1($d))
    $c = $c << ( count1($d) - count1($c) );

    return ($c^$d);
}

    // Driver Code
    $a = 13;
    $b = 5;
    echo XOR1($a, $b);

// This code is contributed by mits
?>
```

Output :

```
7
```

Improved By : Mithun Kumar

Source

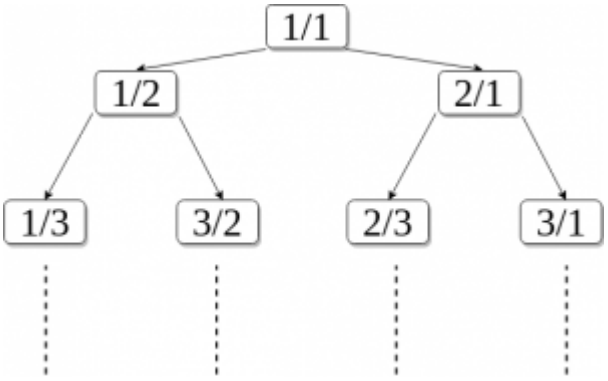
https://www.geeksforgeeks.org/xor-two-numbers-making-length-binary-representations-equal/

Chapter 349

nth Rational number in Calkin-Wilf sequence

nth Rational number in Calkin-Wilf sequence - GeeksforGeeks

What is Calkin Wilf Sequence?
A Calkin-Wilf tree (or sequence) is a special binary tree which is obtained by starting with the fraction 1/1 and adding a/(a+b) and (a+b)/b iteratively below each fraction a/b. This tree generates every rational number. Writing out the terms in a sequence gives 1/1, 1/2, 2/1, 1/3, 3/2, 2/3, 3/1, 1/4, 4/3, 3/5, 5/2, 2/5, 5/3, 3/4, 4/1, …The sequence has the property that each denominator is the next numerator.



The image above is the Calkin-Wilf Tree where all the rational numbers are listed. The children of a node a/b is calculated as a/(a+b) and (a+b)/b.

The task is to find the nth rational number in breadth first traversal of this tree.
Examples:

```
Input : 13
Output : [5, 3]
```

```
Input : 5
Output : [3, 2]
```

Explanation: This tree is a Perfect Binary Search tree and we need floor(log(n)) steps to compute nth rational number. The concept is similar to searching in a binary search tree. Given n we keep dividing it by 2 until we get 0. We return fraction at each stage in the following manner:-

```
    if n%2 == 0
        update frac[1]+=frac[0]
    else
        update frac[0]+=frac[1]
```

Below is the program to find the nth number in Calkin Wilf sequence:

C++

```
 // C++ program to find the
// nth number in Calkin
// Wilf sequence:
# include<bits/stdc++.h>
using namespace std;

int frac[] = {0, 1};

// returns 1x2 int array
// which contains the nth
// rational number
int nthRational(int n)
{
    if (n > 0)
        nthRational(n / 2);

    // ~n&1 is equivalent to
    // !n%2?1:0 and n&1 is
    // equivalent to n%2
    frac[~n & 1] += frac[n & 1];
}

// Driver Code
int main()
{
    int n = 13; // testing for n=13

    // converting array
    // to string format
```

```
    nthRational(n);
    cout << "[" << frac[0] << ","
         << frac[1] << "]" << endl;
    return 0;
}

// This code is contributed
// by Harshit Saini
```

Java

```
 // Java program to find the nth number
// in Calkin Wilf sequence:
import java.util.*;

public class GFG {
    static int[] frac = { 0, 1 };

    public static void main(String args[])
    {
        int n = 13; // testing for n=13

        // converting array to string format
        System.out.println(Arrays.toString(nthRational(n)));
    }

    // returns 1x2 int array which
    // contains the nth rational number
    static int[] nthRational(int n)
    {
        if (n > 0)
            nthRational(n / 2);

        // ~n&1 is equivalent to !n%2?1:0
        // and n&1 is equivalent to n%2
        frac[~n & 1] += frac[n & 1];


        return frac;
    }
}
```

Python3

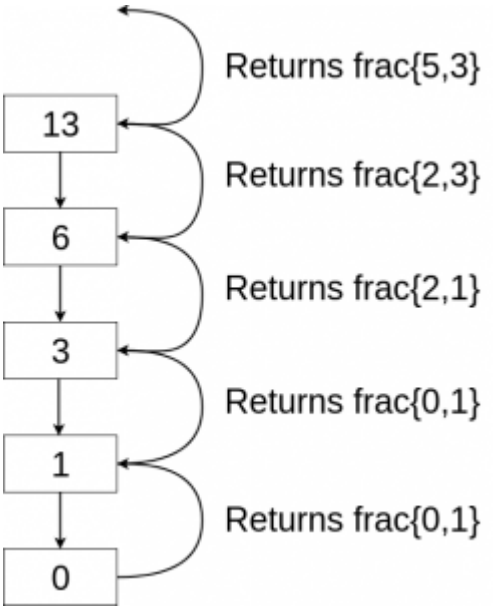
```
 # Python program to find
# the nth number in Calkin
# Wilf sequence:
```

```
frac = [0, 1]

# returns 1x2 int array
# which contains the nth
# rational number
def nthRational(n):
    if n > 0:
        nthRational(int(n / 2))

    # ~n&1 is equivalent to
    # !n%2?1:0 and n&1 is
    # equivalent to n%2
    frac[~n & 1] += frac[n & 1]

    return frac

# Driver code
if __name__ == "__main__":

    n = 13 # testing for n=13

    # converting array
    # to string format
    print(nthRational(n))

# This code is contributed
# by Harshit Saini
```

Output:

```
[5, 3]
```

Explanation:
For n = 13,



Improved By : Harshit Saini

Source

https://www.geeksforgeeks.org/nth-rational-number-in-calkin-wilf-sequence/