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/ 27

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/ 31

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-numbe
r 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

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

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 001100**0**000). 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/ 62

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

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.

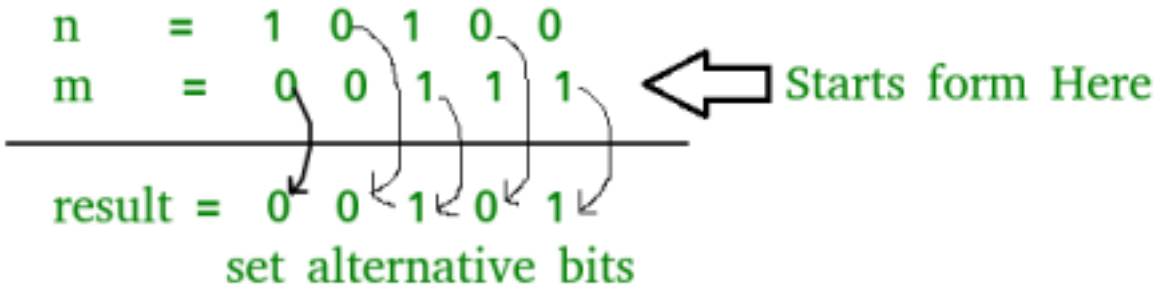
Examples :

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/ 80