

Contents

1 1 to n bit numbers with no consecutive 1s in binary representation 23	Source
 27
2 1 to n bit numbers with no consecutive 1s in binary representation. 28	Source
 31
3 1's and 2's complement of a Binary Number 32	Source
 34
4 A Boolean Array Puzzle 35	Source
	36
5 A backtracking approach to generate n bit Gray Codes 37	Source
 39
6 Add 1 to a given number 40	Source
	46
7 Add minimum number to an array so that the sum becomes even 47	Source
 54
8 Add two bit strings 55	Source 57
9 Add two numbers without using arithmetic operators 58	Source
 62
10 Addition of two numbers without carry 63	Source
 69
11 Alternate bits of two numbers to create a new number 70	Source
 80
12 Baum Sweet Sequence 81	Source
	83
13 Binary representation of a given number 84	Source
 86
1	
<i>Contents</i>	
14 Binary representation of previous number 87	Source
 89
15 Bit Fields in C 90	Source 95

16 Bit Tricks for Competitive Programming 96	Source 99
17 Bitmasking and Dynamic Programming Set-2 (TSP) 100	Source 107
18 Bits manipulation (Important tactics) 108	Source 111
19 Bitwise OR (or) of a range 112	Source 120
20 Bitwise Operators in C/C++ 121	Source 124
21 Bitwise Sieve 125	Source 135
22 Bitwise and (or &) of a range 136	Source 143
23 Bitwise recursive addition of two integers 144	Source 147
24 Bitwise right shift operators in Java 148	Source 149
25 Booth's Multiplication Algorithm 150	Source 159
26 Builtin functions of GCC compiler 160	Source 162
27 C++ bitset and its application 163	Source 167
28 C++ bitset interesting facts 168	Source 171
29 CHAR_BIT in C 172	Source 173
30 Calculate $7n/8$ without using division and multiplication operators 174	Source 180

31 Calculate XOR from 1 to n. 181	Source 186
32 Calculate square of a number without using *, / and pow() 187	Source 196

33 Calculate the total fine to be collected 197	Source	200
34 Case conversion (Lower to Upper and Vice Versa) of a string using BitWise operators in C/C++ 201	Source	203
35 Change all even bits in a number to 0 204	Source	211
36 Change bits to make specific OR value 212	Source	219
37 Check a number is odd or even without modulus operator 220	Source	230
38 Check divisibility in a binary stream 231	Source	234
39 Check divisibility of binary string by 2^k 235	Source	241
40 Check for Integer Overflow 242	Source	244
41 Check if a given number is sparse or not 245	Source	249
42 Check if a number can be expressed as $2^x + 2^y$ 250	Source	258
43 Check if a number can be expressed as a sum of consecutive numbers 259	Source	263
44 Check if a number has bits in alternate pattern Set 1 264	Source	269
45 Check if a number has bits in alternate pattern Set-2 O(1) Approach 270	Source	274
46 Check if a number has same number of set and unset bits 275	Source	278
47 Check if a number has two adjacent set bits 279	Source	282
48 Check if a number is Bleak 283	Source	296
49 Check if a number is divisible by 17 using bitwise operators 297	Source	

.....	302
50 Check if a number is divisible by 8 using bitwise operators 303	Source 306
51 Check if a number is multiple of 9 using bitwise operators 307	Source 311
52 Check if a number is positive, negative or zero using bit operators 312	Source .. 317
53 Check if a number is power of 8 or not 318	Source 320
54 Check if actual binary representation of a number is palindrome 321	Source 327
55 Check if all bits of a number are set 328	Source 337
56 Check if binary representation of a given number and its complement are anagram 338	Source 340
57 Check if binary representation of a number is palindrome 341	Source 342
58 Check if binary representations of two numbers are anagram 343	Source 347
59 Check if binary string multiple of 3 using DFA 348	Source 355
60 Check if bits in range L to R of two numbers are complement of each other or not 356	Source 357
61 Check if bits of a number has count of consecutive set bits in increasing order 360	Source 368
62 Check if bitwise AND of any subset is power of two 369	Source 377
63 Check if concatenation of two strings is balanced or not 378	Source 385

64 Check if given four integers (or sides) make rectangle 386	Source 392
65 Check if given number is a power of d where d is a power of 2 393	Source 399

66 Check if n is divisible by power of 2 without using arithmetic operators 400	
Source	404
67 Check if one of the numbers is one's complement of the other 405	Source
.....	411
68 Check if two numbers are bit rotations of each other or not 412	Source
.....	415
69 Check if two numbers are equal without using arithmetic and comparison operators 416	Source
	419
70 Check if two numbers are equal without using comparison operators 420	Source
.....	425
71 Check in binary array the number represented by a subarray is odd or even 426	Source
	430
72 Check whether K-th bit is set or not 431	Source
.....	437
73 Check whether a given number is even or odd 438	Source
.....	445
74 Check whether all the bits are set in the given range 446	Source
.....	451
75 Check whether all the bits are unset in the given range 452	Source
.....	457
76 Check whether all the bits are unset in the given range or not 458	Source
.....	462
77 Check whether bits are in alternate pattern in the given range 463	Source
.....	470
78 Check whether bits are in alternate pattern in the given range Set-2 471	Source
.....	474
79 Check whether the bit at given position is set or unset 475	Source
.....	479
80 Check whether the number has only first and last bits set 480	5

Contents

Source	484
81 Check whether the number has only first and last bits set Set 2 485	Source
.....	489
82 Check whether the two numbers differ at one bit position only 490	Source
.....	493

83 Closest (or Next) smaller and greater numbers with same number of set bits 494	
Source	519
84 Compare two integers without using any Comparison operator 520	Source
.....	521
85 Compute modulus division by a power-of-2-number 522	Source
.....	525
86 Compute the integer absolute value (abs) without branching 526	Source
.....	528
87 Compute the minimum or maximum of two integers without branching 529	
Source	533
88 Compute the parity of a number using XOR and table look-up 534	Source
.....	538
89 Computing INT_MAX and INT_MIN with Bitwise operations 539	Source
.....	541
90 Construct an array from XOR of all elements of array except element at same index 542	Source
	546
91 Convert a binary number to octal 547	Source
.....	549
92 Convert decimal fraction to binary number 550	Source
.....	553
93 Copy set bits in a range 554	Source
..	556
94 Count all pairs of an array which differ in K bits 557	Source
.....	562
95 Count all pairs with given XOR 563	Source
.....	567
96 Count inversions in an array Set 3 (Using BIT) 568	Source
.....	573
97 Count minimum bits to flip such that XOR of A and B equal to C 574	Source
.....	579
98 Count number of bits to be flipped to convert A to B 580	Source
.....	585
99 Count number of distinct sum subsets within given range 586	Source

.....	588
100 Count number of subsets having a particular XOR value 589 Source	592
101 Count numbers whose sum with x is equal to XOR with x 593 Source	600
102 Count of divisors having more set bits than quotient on dividing N 601 Source .	610
103 Count pairs in an array which have at least one digit common 611 Source	617
104 Count pairs with Bitwise AND as ODD number 618 Source	626
105 Count pairs with Bitwise OR as Even number 627 Source	633
106 Count pairs with Bitwise XOR as ODD number 634 Source	642
107 Count pairs with Odd XOR 643 Source	651
108 Count set bits in a range 652 Source	657
109 Count set bits in an integer 658 Source	674
110 Count set bits in an integer using Lookup Table 675 Source	677
111 Count set bits using Python List comprehension 678 Source	679
112 Count smaller numbers whose XOR with n produces greater value 680 Source .	686
113 Count smaller values whose XOR with x is greater than x 687 Source	695

114 Count strings with consecutive 1's 696 Source	701
115 Count total bits in a number 702 Source	708

116 Count total set bits in all numbers from 1 to n 709	Source	721
117 Count trailing zero bits using lookup table 722	Source	728
118 Count unset bits in a range 729	Source	734
119 Count unset bits of a number 735	Source	740
120 Cyclic Redundancy Check and Modulo-2 Division 741	Source	744
121 Decimal Equivalent of Gray Code and its Inverse 745	Source	752
122 Decimal representation of given binary string is divisible by 10 or not 753	Source	761
123 Decimal representation of given binary string is divisible by 20 or not 762	Source	771
124 Detect if two integers have opposite signs 772	Source	775
125 Determine if a string has all Unique Characters 776	Source	789
126 Disjoint Set Union on trees Set 1 790	Source	793
127 Disjoint Set Union on trees Set 2 794	Source	798
128 Divide two integers without using multiplication, division and mod operator 799	Source	810
129 Divisibility by 64 with removal of bits allowed 811	Source	815
130 Efficient method for 2's complement of a binary string 816	Source	820
131 Efficient way to multiply with 7 821	Source	824
132 Efficiently check if a string has duplicates without using any additional data		

structure 825	Source	830
133 Efficiently check whether n is a multiple of 4 or not 831	Source	
.		834
134 Efficiently find first repeated character in a string without using any additional data structure in one traversal 835	Source	
.		839
135 Equal Sum and XOR 840	Source	
.		848
136 Euclid's Algorithm when % and / operations are costly 849	Source	
.		850
137 Extract 'k' bits from a given position in a number. 851	Source	
.		854
138 Fast average of two numbers without division 855	Source	
.		857
139 Fast inverse square root 858	Source	
.		860
140 Fibbinary Numbers (No consecutive 1s in binary) 861	Source	
.		866
141 Fibbinary Numbers (No consecutive 1s in binary) – O(1) Approach 867	Source	
.		871
142 Find Duplicates of array using bit array 872	Source	
.		874
143 Find Next Sparse Number 875	Source	
.		878
144 Find One's Complement of an Integer 879	Source	
.		882
145 Find Two Missing Numbers Set 2 (XOR based solution) 883	Source	
.		892
146 Find Unique pair in an array with pairs of numbers 893	Source	
.		899
147 Find XOR of two number without using XOR operator 900		9

Contents

Source	904
148 Find even occurring elements in an array of limited range 905	Source
.	908

149 Find i'th index character in a binary string obtained after n iterations Set 2	909
Source	917
150 Find largest element from array without using conditional operator	918
Source	921
151 Find longest sequence of 1's in binary representation with one flip	922
Source	927
152 Find missing number in another array which is shuffled copy	928
Source	933
153 Find most significant set bit of a number	934
Source	946
154 Find nth Magic Number	947
Source	952
155 Find number of pairs in an array such that their XOR is 0	953
Source	962
156 Find one extra character in a string	963
Source	969
157 Find position of the only set bit	970
Source	980
158 Find profession in a special family	981
Source	989
159 Find smallest number n such that n XOR n+1 equals to given k.	990
Source	994
160 Find the Number Occurring Odd Number of Times	995
Source	1004
161 Find the element that appears once	1005
Source	1019
162 Find the largest number with n set and m unset bits	1020
Source	1025
163 Find the maximum subarray XOR in a given array	1026
Source	1036

164 Find the maximum subset XOR of a given set	1037
Source	1047

165 Find the missing element in an array of integers represented in binary format	1048	Source	1052
166 Find the n-th number whose binary representation is a palindrome	1053	Source	1063
167 Find the smallest number with n set and m unset bits	1064	Source	1069
168 Find the two non-repeating elements in an array of repeating elements	1070	Source	1072
169 Find the winner in nim-game	1073	Source	1078
170 Find two numbers from their sum and XOR	1079	Source	1084
171 Find value of k-th bit in binary representation	1085	Source	1088
172 Find whether a given number is a power of 4 or not	1089	Source	1098
173 Finding the Parity of a number Efficiently	1099	Source	1104
174 First element greater than or equal to X in prefix sum of N numbers using Binary Lifting	1105	Source	1107
175 For every set bit of a number toggle bits of other	1108	Source	1111
176 Game of Nim with removal of one stone allowed	1112	Source	1115
177 Generate 0 and 1 with 25% and 75% probability	1116	Source	1121
178 Generate n-bit Gray Codes	1122	Source	1124
179 Generate n-bit Gray Codes Set 2	1125	Source	1130
180 Get the position of rightmost unset bit	1131	11	

Contents

Source	1136
181 Given a set, find XOR of the XOR's of all subsets.	1137
Source	1141

182 Gray to Binary and Binary to Gray conversion 1142	Source
1149
183 Highest power of 2 less than or equal to given number 1150	Source
1160
184 How to swap two bits in a given integer? 1161	Source
1162
185 How to swap two numbers without using a temporary variable? 1163	Source
1174
186 How to turn off a particular bit in a number? 1175	Source
1179
187 How to turn on a particular bit in a number? 1180	Source
1184
188 Increment a number without using ++ or + 1185	Source
1191
189 Increment a number by one by manipulating the bits 1192	Source
1199
190 Inserting M into N such that m starts at bit j and ends at bit i Set-21200	Source
1208
191 Inserting m into n such that m starts at bit j and ends at bit i. 1209	Source
1215
192 Invert actual bits of a number 1216	Source
1221
193 Josephus Problem Using Bit Magic 1222	Source
1229
194 Josephus problem Set 1 (A O(n) Solution) 1230	Source
1234
195 Largest number with binary representation is m 1's and m-1 0's 1235	Source
1241
196 Largest set with bitwise OR equal to n 1242	Source
1243
197 Left Shift and Right Shift Operators in C/C++ 1244	12

Contents

Source	1246
------------------	------

198 Leftover element after performing alternate Bitwise OR and Bitwise XOR

operations on adjacent pairs 1247	Source	1260
199 Length of the Longest Consecutive 1s in Binary Representation 1261	Source	1266
200 Levelwise Alternating OR and XOR operations in Segment Tree 1267	Source	1276
201 Little and Big Endian Mystery 1277	Source	1280
202 Lucky alive person in a circle Code Solution to sword puzzle 1281	Source	1283
203 M-th smallest number having k number of set bits. 1284	Source	1286
204 Maximize a given unsigned number number by swapping bits at it's extreme positions. 1287	Source	1289
205 Maximize the bitwise OR of an array 1290	Source	1296
206 Maximize the number by rearranging bits 1297	Source	1307
207 Maximum 0's between two immediate 1's in binary representation 1308	Source	1312
208 Maximum AND value of a pair in an array 1313	Source	1322
209 Maximum OR sum of sub-arrays of two different arrays 1323	Source	1327
210 Maximum XOR using K numbers from 1 to n 1328	Source	1332
211 Maximum XOR value of a pair from a range 1333	Source	1339
212 Maximum XOR-value of at-most k-elements from 1 to n 1340	Source	1343
213 Maximum set bit sum in array without considering adjacent elements 1344	Source	1350
214 Maximum steps to transform 0 to X with bitwise AND 1351	Source	1355

215 Maximum subset with bitwise OR equal to k 1356	Source
1363
216 Maximum sum by adding numbers with same number of set bits 1364	Source
1370
217 Minimum bit changes in Binary Circular array to reach a index 1371	Source
1374
218 Minimum bitwise operations to convert given a into b. 1375	Source
1377
219 Minimum digits to remove to make a number Perfect Square 1378	Source
1387
220 Minimum flips required to maximize a number with k set bits 1388	Source
1394
221 Minimum flips to make all 1s in left and 0s in right Set 1 (Using Bitmask) 1395	Source
1397
222 Minimum number using set bits of a given number 1398	Source
1401
223 Minimum value of N such that xor from 1 to N is equal to K 1402	Source
1407
224 Modify a bit at a given position 1408	Source
1411
225 Multiples of 4 (An Interesting Method) 1412	Source
1419
226 Multiplication of two numbers with shift operator 1420	Source
1424
227 Multiplication with a power of 2 1425	Source
1431
228 Multiply a given Integer with 3.5 1432	Source
1435
229 Multiply a number with 10 without using multiplication operator 1436	Source
1439
230 Multiply any Number with 4 using Bitwise Operator 1440	Source
1443

1444 Source	1447
232 Next greater integer having one more number of set bits 1448 Source	1453
233 Next higher number with same number of set bits 1454 Source	1458
234 Number of Reflexive Relations on a Set 1459 Source	1462
235 Number of integers with odd number of set bits 1463 Source	1467
236 Number of pairs with Pandigital Concatenation 1468 Source	1477
237 Number of unique triplets whose XOR is zero 1478 Source	1483
238 Number whose XOR sum with given array is a given number k 1484 Source	1488
239 Number whose sum of XOR with given array range is maximum 1489 Source	1499
240 Number with set bits only between L-th and R-th index 1500 Source	1507
241 Numbers whose bitwise OR and sum with N are equal 1508 Source	1514
242 Odd numbers in N-th row of Pascal's Triangle 1515 Source	1521
243 Odious number 1522 Source	1527
244 Operators in C Set 2 (Relational and Logical Operators) 1528 Source	1532
245 Optimization Techniques Set 1 (Modulus) 1533 Source	1534
246 Pairs of complete strings in two sets of strings 1535 Source	1539
247 Pairs whose concatenation contain all digits 1540 Source	1542

248 Pernicious number 1543 Source	
---	--

	.1549
249 Position of rightmost bit with first carry in sum of two binary 1550	Source1553
250 Position of rightmost common bit in two numbers 1554	Source1558
251 Position of rightmost different bit 1559	Source1564
252 Position of rightmost set bit 1565	Source1573
253 Powers of 2 to required sum 1574	Source1580
254 Previous number same as 1's complement 1581	Source1584
255 Previous smaller integer having one less number of set bits 1585	Source1589
256 Prime Number of Set Bits in Binary Representation Set 1 1590	Source1592
257 Print all subsequences of a string Iterative Method 1593	Source1598
258 Print all the combinations of N elements by changing sign such that their sum is divisible by M 1599	Source1606
259 Print bitwise AND set of a number N 1607	Source1613
260 Print first n numbers with exactly two set bits 1614	Source1619
261 Print numbers having first and last bits as the only set bits 1620	Source1630
262 Print numbers in the range 1 to n having bits in alternate pattern 1631	Source1638
263 Print pair with maximum AND value in an array 1639	Source1648
264 Print 'K'th least significant bit of a number 1649	Source1652

265 Program to count number of set bits in an (big) array 1653	Source
1656
266 Program to find parity 1657	Source
	..1661
267 Program to find whether a no is power of two 1662	Source
1671
268 Program to invert bits of a number Efficiently 1672	Source
1674
269 Python Slicing Extract 'k' bits from a given position 1675	Source
1676
270 Python map function Count total set bits in all numbers from 1 to n1677	
	Source
1678
271 Python program to convert floating to binary 1679	Source
1681
272 Python Count set bits in a range 1682	Source
1683
273 Python Count unset bits in a range 1684	Source
1685
274 Queries for number of array elements in a range with Kth Bit Set 1686	Source ..
1693
275 Queries on XOR of XORs of all subarrays 1694	Source
1702
276 Quotient and remainder dividing by 2^k (a power of 2) 1703	Source
1707
277 Range query for count of set bits 1708	Source
1713
278 Remove duplicates from a string in $O(1)$ extra space 1714	Source
1717
279 Reverse actual bits of the given number 1718	Source
1723
280 Reverse an array without using subtract sign '-' anywhere in the code1724	
	Source
1726
281 Reverse bits using lookup table in $O(1)$ time 1727	Source
1729

282 Rotate bits of a number 1730	Source	1734
283 Russian Peasant (Multiply two numbers using bitwise operators) 1735	Source	1740
284 Same Number Of Set Bits As N 1741	Source	1746
285 Set all even bits of a number 1747	Source	1758
286 Set all odd bits of a number 1759	Source	1770
287 Set all the bits in given range of a number 1771	Source	1775
288 Set bits in N equals to M in the given range. 1776	Source	1781
289 Set the K-th bit of a given number 1782	Source	1785
290 Set the Left most unset bit 1786	Source	1791
291 Set the rightmost unset bit 1792	Source	1795
292 Shuffle a pack of cards and answer the query 1796	Source	1800
293 Smallest number whose set bits are maximum in a given range 1801	Source	1808
294 Smallest of three integers without comparison operators 1809	Source	1813
295 Smallest perfect power of 2 greater than n (without using arithmetic operators) 1814	Source	1818
296 Smallest power of 2 greater than or equal to n 1819	Source	1831
297 Space optimization using bit manipulations 1832	Source	1838

298 String transformation using XOR and OR 1839	Source
.....1846	

299 Subset sum queries using bitset 1847	Source
.....1850	

300 Subtract 1 without arithmetic operators 1851	Source
.....1855	

301 Subtract two numbers without using arithmetic operators 1856	Source
.....1864	

302 Sudo Placement Range Queries 1865	Source
.....1870	

303 Sum of Bitwise And of all pairs in a given array 1871	Source
.....1880	

304 Sum of XOR of all pairs in an array 1881	Source
.....1891	

305 Sum of XOR of sum of all pairs in an array 1892	Source
.....1898	

306 Sum of all elements up to Nth row in a Pascal triangle 1899	Source
.....1907	

307 Sum of bit differences among all pairs 1908	Source
.....1913	

308 Sum of bitwise AND of all possible subsets of given set 1914	Source
.....1921	

309 Sum of bitwise OR of all possible subsets of given set 1922	Source
.....1927	

310 Sum of bitwise OR of all subarrays 1928	Source
.....1935	

311 Sum of numbers with exactly 2 bits set 1936	Source
.....1944	

312 Sum of the series $2^0 + 2^1 + 2^2 + \dots + 2^n$ 1945	Source
.....1952	

313 Swap all odd and even bits 1953	Source
.....1957	

314 Swap bits in a given number 1958	Source
---	--------------

.....	1963
315 Swap every two bits in bytes 1964	Source
.....	1967
19	
<i>Contents</i>	
316 Swap three variables without using temporary variable 1968	Source
.....	1974
317 Swap two nibbles in a byte 1975	Source
.....	1978
318 Toggle all bits after most significant bit 1979	Source
.....	1988
319 Toggle all even bits of a number 1989	Source
.....	1993
320 Toggle all odd bits of a number 1994	Source
.....	1998
321 Toggle all the bits of a number except k-th bit. 1999	Source
.....	2000
322 Toggle bits in the given range 2001	Source
.....	2005
323 Toggle bits of a number except first and last bits 2006	Source
.....	2011
324 Toggle case of a string using Bitwise Operators 2012	Source
.....	2015
325 Toggle first and last bits of a number 2016	Source
.....	2021
326 Toggle the last m bits 2022	Source
.....	2026
327 Toggling k-th bit of a number 2027	Source
.....	2030
328 Travelling Salesman Problem Set 1 (Naive and Dynamic Programming)2031	
Source	2032
329 Turn off the rightmost set bit 2033	Source
.....	2036
330 Two odd occurring elements in an array where all other occur even times 2037	
Source	2042

331 Unique element in an array where all elements occur k times except one	2043
Source	2045
332 Unset bits in the given range	2046
Source	2051
333 Unset the last m bits	2052
Source	2055
334 Value in a given range with maximum XOR	2056
Source	2059
335 Variation in Nim Game	2060
Source	2067
336 Ways to represent a number as a sum of 1's and 2's	2068
Source	2070
337 Ways to split array into two groups of same XOR value	2071
Source	2075
338 What are the differences between bitwise and logical AND operators in C/C++?	2076
Source	2078
339 Write a function that returns 2 for input 1 and returns 1 for 2	2079
Source	2080
340 Write an Efficient C Program to Reverse Bits of a Number	2081
Source	2083
341 Write an Efficient Method to Check if a Number is Multiple of 3	2084
Source	2091
342 Write your own strcmp that ignores cases	2092
Source	2093
343 XNOR of two numbers	2094
Source	2108
344 XOR counts of 0s and 1s in binary representation	2109
Source	2112
345 XOR Encryption by Shifting Plaintext	2113
Source	2116
346 XOR of Sum of every possible pair of an array	2117
Source	2121
347 XOR of all subarray XORs Set 2	2122
Source	

.....2126

348 XOR of two numbers after making length of their binary representations equal
2127 Source2134

349 nth Rational number in Calkin-Wilf sequence 2135 21

Contents

Source2139 22

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
        // common 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
            // common 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 then 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](#)

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 devendraiit 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 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
```

40

Chapter 6. Add 1 to a given number

```
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));
    }
}
```

41

Chapter 6. Add 1 to a given number

// 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 Purna 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;
        }
    }
}

```

42

Chapter 6. Add 1 to a given number

```

// 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

$\sim x = -(x+1)$ [\sim 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 $(-(\sim x))$.

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),
 $\sim x = \sim 2 = 1101$ (13 numerical)
 $\sim \sim x = -1101$

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;
    }
}

```

48

Chapter 7. Add minimum number to an array so that the sum becomes even

```

}

// 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)  
    {
```

49

Chapter 7. Add minimum number to an array so that the sum becomes even

```
        // 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.

```

50

Chapter 7. Add minimum number to an array so that the sum becomes even

?>

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
```

51

Chapter 7. Add minimum number to an array so that the sum becomes even

```
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 beginning 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 adding leading 0s in the smaller string. Returns the // the new length
int makeEqualLength(string &str1, string &str2)
{
```

55

Chapter 8. Add two bit strings

```
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;
```

58

Chapter 9. Add two numbers without using arithmetic operators

```
        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

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.

63

Chapter 10. Addition of two numbers without carry

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;
    }
}

```

64

Chapter 10. Addition of two numbers without carry

```

    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

```

66

Chapter 10. Addition of two numbers without carry

```

        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

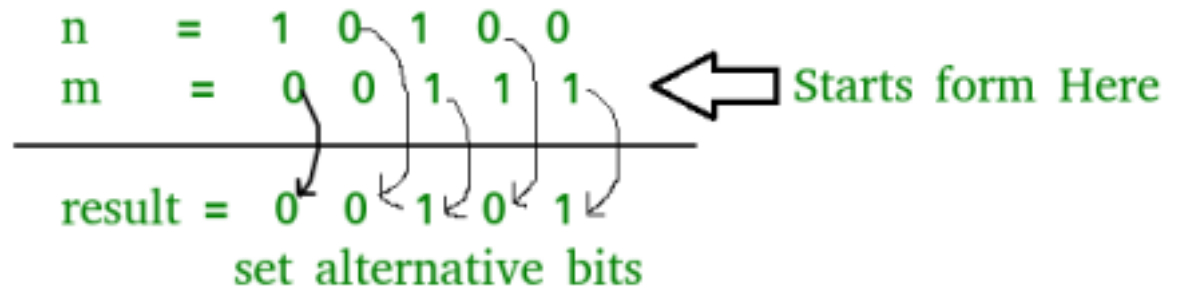
Examples :

Output is = 1 0 1 1

Output is = 0 0 1 0 1

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));
}
}

```

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.....
    }
}

```

Chapter 11. Alternate bits of two numbers to create a new number

```
// 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
    $tempn = setevenuebits($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
// ^ ^
```

79

Chapter 11. Alternate bits of two numbers to create a new number

```
// 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

