

3. Bit Wise

AND (2) OR (1) XOR (1)

AND:

1st	2nd	res
0	0	0
0	1	0
1	0	0
1	1	1

OR:

1st	2nd	res
0	0	0
0	1	1
1	0	1
1	1	1

XOR:

1st	2nd	res
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise not (~)

X: 1011
~X: 0100

1st	2nd	res
1	0	
0	1	

For n bits, (unsigned)
Min: 0
Max: $2^n - 1$
ex: n=4,
Min: 0, Max: $2^4 - 1 = 15$
(1111)

Left Shift (<<)

Right Shift (>>)

Left shift: * Bits moves left side

ex: x: 00000110

discarded 00001100
zero is added at end

We shift only 1 bit so only 1 zero is added

* Multiplies the given num by 2^n where n is no. of shifts.

ex: 5 = 0101 & let n=1

x 0101
1010 (5 x 2 = 10)
zero added

Right shift: * Bits moves Right side

ex: x = 00101000

discarded
zero added 001010

* Divides the given num by 2^n where n is no. of shifts.

ex: 10: 1010, n=1.

1010 discarded
0101
5 (10 / 2 = 5)

3 & 5, 5: 101
3: 011
1: 001

3 | 5, 5: 101
3: 011
7: 111

3 ^ 5, 5: 101
3: 011
6: 110

For n bits, (signed)
Min: -2^{n-1}
Max: $2^{n-1} - 1$

ex: 25 comp of x in n bits res = $2^n - x$

ex: n=3, $2^3 - 2 = 8 - 2 = 6$
2 in 3 bits is (110)

-ve signed: stored in 2's complement representation

0001 +ve 1000 -ve

22: 10110, n=2, Left shift & Right shift

22: 0010110
discarded 0010110
1011000 - 88

22 x 2 = 44
22 x 4 = 88
no. of shifts

discarded 2 shifts
11000
2 zeros added
22 x 2 = 44
22 x 4 = 88

1/ unsigned Input (0 to $2^n - 1$)

Bitwise AND

$x: 000 \dots 01$ (32 bit rep of 1)

$\neg x: (1)1 \dots 10$

signed
-ve number

$(2^{32} - 1) = 2^{32} / 2$

$x: 000 \dots 0101$

$\neg x: (1)1 \dots 1010$

signed

$2^{32} - 1 - 5 = 2^{32} / 6$

$2^{32} - 1$
 $= 8 - 1 = 7$

2) Count Set Bits :

c) $I(p) \quad n=5, \quad o/p = 2 \leftarrow$

↳ B.R = 0101 2 bits or 1

```
int countBits (int n) {
    int res=0;
    while(n>0){
        if(n&1) res++;
        n = n>>1;
    }
    return res;
}
```

$$\frac{101}{1} \rightarrow \text{True} +1$$

$$\frac{1010}{10} \rightarrow \text{false}$$

$$\frac{10}{1} \rightarrow \text{true} +1$$

```
int countBits (int n) {
    int res = 0;
    while (n > 0) {
        n = (n & (n-1));
        res++;
    }
    return res;
}
```

BRIAN
KERNINGMAN'S
Algorithm
 $\Theta(n \log \text{set bits})$

$n = 5$
 $5 - 101$
 $4 - 102$
 $\underline{100}$
 $808 = +1$

$4 - 100$
 $3 - 011$
 $\underline{000}$
 $208 = +1$

$0 \times 0 = 0$
 $808 = 2$

$n=4$

$$\begin{array}{r} 7-111 \\ 6-110 \\ \hline 110 \\ +1 \end{array} \rightarrow \begin{array}{r} 6-110 \\ 5-101 \\ \hline 100 \\ +1 \end{array} \rightarrow \begin{array}{r} 4-100 \\ 3-011 \\ \hline 000 \\ +1 \end{array} \rightarrow \text{000f}$$

The final result is circled and labeled **cell-3**.

$n = 11$
 $11 \rightarrow 1011$
 $10 \rightarrow 1010$
 $f1$

1011
 1010
 1010
 $+1$

1011
 1010
 1000
 $+1$

841000
 $7 \rightarrow 1011$
 0000
 71

$070 F$
 $221 = 3$

$n: 000 \dots 1101$ 13 > 2
 $n \gg (x-1): 000 \dots 001$
 kth bit moves to
 least bit position & checks

Lookup Table method for 32 bit numbers for Count Bits :-

$n = 13$

$000 \dots 0$ 8 bits 4th
 $00 \dots 0$ 8 bits 3rd
 $0 \dots 0$ 8 bits 2nd
 $0 \dots 1101$ 8 bits 1st block

total 32 bit numbers
4 blocks of 8 bits

```

int table[256];
void initialize()
{
    table[0] = 0;
    for (int i = 1; i < 256; i++) {
        table[i] = (i & 1) + table[i/2];
    }
}
    
```

for $i = 5$, 101 \rightarrow Test 1 \rightarrow +2
 $table[i/2] + table[i] = 4$

3
int count(int n)

```

{
    int res = table[n & 0xff]; // Hexa Decimal number of 8
    n = n >> 8; // right shift to 8 bits
    res = res + table[n & 0xff];
    n = n >> 8;
    res = res + table[n & 0xff];
    n = n >> 8;
    res = res + table[n & 0xff];
    return res;
}
    
```

If 64 bits then 8 blocks are present

3) Power of Two :-

i) I/p $n = 4$, o/p = Yes
 ii) I/p $n = 6$, o/p = No

11y, $\frac{4}{2} \Rightarrow \frac{2}{2} \Rightarrow \frac{1}{2} \Rightarrow \frac{1}{2}$ Yes
 $\frac{6}{2} \Rightarrow \frac{3}{2} \Rightarrow \frac{3}{2}$ No

Nave:

```

bool isPower2(int n)
{
    if (n == 0)
        return false;
    while (n != 1) {
        if (n % 2 != 0)
            return false;
        n = n / 2;
    }
    return true;
}
    
```

$\therefore O(\log n)$

Optimal

$2^0 \Rightarrow 1 \Rightarrow 000 \dots 0001$
 $2^1 \Rightarrow 2 \Rightarrow 000 \dots 0010$
 $2^2 \Rightarrow 4 \Rightarrow 000 \dots 0100$
 $2^3 \Rightarrow 8 \Rightarrow 000 \dots 1000$

The no of set bits for the value (2^n) is always equal to 1.
 Use Brian Kernighan's Algo to count set bits.

Method 2:-

```

bool isPower2(int n)
{
    if (n == 0)
        return false;
    return (n & (n-1)) == 0;
}
    
```

$4 \Rightarrow 1000$
 $3 \Rightarrow 011$
 $4 \& 3 = 0100 \& 0011 = 0000$

4) Find the only odd occurring number:-

ex: $arr = [4, 3, 4, 4, 4, 5, 5]$

o/p = 3

$\begin{matrix} 4 \rightarrow 4 \\ 3 \rightarrow 1 \\ 5 \rightarrow 2 \end{matrix}$

Here, odd occurring, even occurring : use XOR bitwise

In XOR, $x \wedge 0 = x$

$x \wedge x = 0$

$x \wedge y = y \wedge x$ (Commutative)

$x \wedge (y \wedge z) = (x \wedge y) \wedge z$ (Associative)

```

int findOdd(int arr[], int n)
{
    int res = 0;
    for (int i = 0; i < n; i++) {
        res = res ^ arr[i];
    }
    return res;
}
    
```

TC: $O(n)$

SC: $O(1)$ Aux Space

Variation Q/A:- Given an array of n numbers that has values in range $[1 \dots n+1]$. Every no. appears exactly once. Hence one number is missing. Find the missing number?

Ex: $[1, 4, 3]$
o/p = 2

(1 to $n+1$) numbers $\rightarrow (1 \wedge 2 \wedge 3 \wedge 4) \wedge (1 \wedge 4 \wedge n \wedge 3)$
 $n=4$

\rightarrow only 2 comes out as other numbers occur 2 times

5) Find the two odd appearing numbers:-

Ex: $arr = [3, 4, 3, 4, 5, 4, 4, 6, 7, 4]$

o/p $\{5, 6\}$

```

void oddAppearing(int arr[], int n)
{
    int xor = 0, res1 = 0, res2 = 0;
    for (int i = 0; i < n; i++) { xor = xor ^ arr[i]; }
    int rn = (xor) & (~ (xor - 1)) // Right Most Bit set of xor
    for (int i = 0; i < n; i++)
    {
        if ((arr[i] & rn) != 0)
            res1 = res1 ^ arr[i];
        else
            res2 = res2 ^ arr[i];
    }
    return {res1, res2};
}
    
```

$O(n)$: TC

$O(1)$: SC

$5 \rightarrow 101$
 $6 \rightarrow 110$
 $xor = 5 \wedge 6 = \dots 011 \rightarrow 3$
 $(xor - 1) \rightarrow \dots 010 \rightarrow 2$
 $\sim(xor - 1) = \dots 101$
 $(xor) \& (\sim(xor - 1)) = 1$

$3 \rightarrow 011$
 $4 \rightarrow 100$
 $5 \rightarrow 101$
 $6 \rightarrow 110$
 $7 \rightarrow 111$
 $4 \rightarrow 100$
 $4 \rightarrow 100$
 $4 \rightarrow 100$
 $4 \rightarrow 100$
 $6 \rightarrow 110$
 $7 \rightarrow 111$

$res1 = (3 \wedge 5 \wedge 7 \wedge 4 \wedge 6) \rightarrow 5$
 $res2 = (4 \wedge 4 \wedge 4 \wedge 4 \wedge 4) \rightarrow 6$
 $\{5, 6\}$

6) Power Set using Bitwise Operators :-

Input: $s = "abc"$

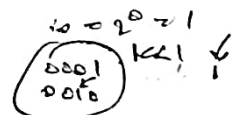
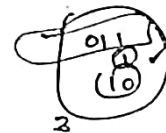
Output: "", "a", "b", "c", "ab", "ac", "bc", "abc"

for 'n' character we have 2^n subsets $n=3, 2^3=8$

$n=3, 2^3-1=8-1=7$ sets 0 to 7 decimal.

Counter (Decimal)	Counter (Binary)	Subset
0	000	"
1	001	"a"
2	010	"b"
3	011	"ab"
4	100	"c"
5	101	"ac"
6	110	"bc"
7	111	"abc"

```
void printPowerSet(string str)
{
    int n = str.length(); // n=3
    int powersize = pow(2, n); // 8
    for (int c = 0; c < powersize; c++)
    {
        for (int s = 0; s < n; s++) // abc
        {
            if ((c & (1 << s)) != 0)
                print(str[s]);
        }
        print("\n");
    }
}
```



1 & (1 << 0)
 $1 \times 2^0 = 1 \times 1$
 $1 \times 1 = 1 \rightarrow a$
 2 & (1 << 0)
 $1 \times 2^0 = 1 \times 1$

