Bit Manipulation - I

Prerequisites: knowledge of binary number system

Get i^{th} bit:

Mask: Right shift n 'i' times, and check the first bit.

```
int getBit(int n, int pos){
    return (n >> pos) & 1;
}
```

Set i^{th} bit:

Mask: 1 << i

Bitwise OR operation between n and mask sets the i^{th} bit to one.

```
int setBit(int n, int pos) {
    return (n | (1 << pos));
}
```

Clear i^{th} bit

Mask: ~ (1 << i)

In the mask, all the bits would be one, except the i^{th} bit. Taking bitwise AND with n would clear the i^{th} bit.

```
int clearBit(int n, int pos) {
    int mask = ~(1 << pos);
    return (n & mask);
}
```

Toggle i^{th} bit

Mask: 1 << i

Bitwise XOR operation between n and mask toggle the i^{th} bit.

```
int toggleBit(int n, int pos) {
    return (n xor (1 << pos));
}
```

Update i'th bit to the given value

Mask: mask has all the bits set to one except i^{th} bit.

n = n & mask, i^{th} bit is cleared.

Now, to set i^{th} bit to value, we take value << pos as the mask.

```
int updateBit(int n, int pos, int value) {
    int mask = ~(1 << pos);
    n = n & mask;
    return (n | (value << pos));
}
```

Supplementary material

Compute XOR from 1 to n (direct method) :

```
int computeXOR(int n)
{
    if (n % 4 == 0)
        return n;
    if (n % 4 == 1)
        return 1;
    if (n % 4 == 2)
        return n + 1;
    else
        return 0;
}
```

Input: 6

Output: 7

Equal Sum and XOR

Problem: Given a positive integer n, find count of positive integers i such that 0 <= i <= n and n+i = n⊕i (⊕ is the XOR operation)

Instead of using looping (Brute force method), we can directly find it by a mathematical trick i.e.

Let x be the number of unset bits in the number n.

Answer = 2^x

XOR of all subsequences of an array

The answer is always 0 if the given array has more than one element. For an array with a single element, the answer is the value of the single element.

Logic: If the array has more than one element, then element occurs.

Number of leading zeros, trailing zeroes and number of 1's of a number

It can be done by using inbuilt function i.e.

Number of leading zeroes: *builtin_clz(x)*

Number of trailing zeroes: *builtin_ctz(x)*
Number of 1-bits: *__builtin_popcount(x)*

Convert binary numbers directly into a decimal integer in C++.

```
#include <iostream>
using namespace std;
 int main()
{
   int number = 0b011;
   cout << number;
   return 0;
}
```

Output: 3

Swap 2 numbers using bit operations:

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

Flip the bits of a number:

It can be done by a simple way, just simply subtract the number from the value obtained when all the bits are equal to 1 .

For example:

Number: Given Number

Value : A number with all bits set in a given number.

Flipped number = Value – Number.

Example :

Number = 23,

Binary form: 10111

After flipping digits number will be: 01000

Value: 11111 = 31

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

```
int setBitNumber(int n)
{
   n |= n>>1;

   n |= n>>2;
    n |= n>>4;
   n |= n>>8;
   n |= n>>16;

   n = n + 1;
    return (n >> 1);
}
```

Practice questions:

1. Reverse bits
2. Hamming distance