Bit Manipulation - I

Prerequisites: knowledge of binary number system

Get ith bit:

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

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

Set ith bit:

Mask: 1 << i

Bitwise OR operation between n and mask sets the ith bit to one.

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

Clear ith bit

Mask: ~ (1 << i)

In the mask, all the bits would be one, except the ith bit. Taking bitwise AND with n would clear the ith bit.

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

Apni Kaksha

Toggle ith bit

Mask: 1 << i

Bitwise XOR operation between n and mask toggle the ith bit.

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

Update i'th bit to the given value

Mask: mask has all the bits set to one except ith bit. n = n & mask, ith bit is cleared.

Now, to set ith 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));
}</pre>
```

Apmi Kaksha

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 $0 \le i \le n$ and $n+i = n \oplus i$ (\oplus 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;
}</pre>
```

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. <u>Hamming distance</u>

Bit Manipulation - II

Prerequisites: knowledge of binary number system

Count set bits

```
n & (n - 1) sets the first set-bit to zero.

Explanation: n = XXX100

n - 1 = XXX011

n & (n - 1) = XXX000
```

```
int numberofones(int n) {
   int count = 0;
   while (n) {
        n = n & (n - 1);
        count++;
   }
   return count;
}
```

Power of two

From our past knowledge of the binary number system, Numbers of the type 2ⁿ have only 1 set bit.

```
Explanation: n = 000100

n - 1 = 000011

n & (n - 1) = 000000

!( n & (n - 1)) = 000001
```

If the number only had one set bit, then n & (n - 1) would be zero.

```
bool ispowerof2(int n) {
    return (n && !(n & n - 1));
}
```

Generate Subset

Explanation: if the jth bit is set, then we take the jth element.

There are a total of 2ⁿ subsets.

```
void subsets(int arr[], int n) {
    for (int i = 0; i < (1 << n); i++) {
        for (int j = 0; j < n; j++) {
            if ( i & (1 << j)) {
                cout << arr[j] << " ";
            }
        } cout << endl;
}</pre>
```

Practice Questions:

- 1. Counting bits
- 2. Power of four

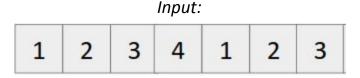
Bit Manipulation Challenges

Challenge 1

Write a program to find a unique number in an array where all numbers except one, are present twice.

Hint: $A \oplus B \oplus B \oplus A \oplus C = C$. All those numbers which occur twice will get nullified after \oplus operation.

Sample Test Case:



Output: 4

Code

```
#include<iostream>
using namespace std;
int unique(int arr[], int n) {
    int xorsum = 0;
    for (int i = 0; i < n; i++) {
        xorsum = xorsum ^ arr[i];
    }
    return xorsum;
}
int main() {
    int arr[] = {1, 2, 3, 4, 1, 2, 3};
    cout << unique(arr, 7) << endl;
    return 0;
}</pre>
```

Apmi Kaksha

Challenge 2

Q2. Write a program to find 2 unique numbers in an array where all numbers except two, are present twice.

Logic

- 1. Take XOR of all the elements and let that xor value be x. All the repeating elements will get nullified and xor of only two unique elements will last. (as $a \oplus a = 0$).
- 2. There will be at least one bit set in x. Using that set bit, divide the original set of numbers into two sets
 - a. First set which contains all the elements with that bit set.
 - b. Second set which contains all the elements with that bit unset.
- 3. Take xor of both the sets individually, and let those xor values be x1 and x2.
- 4. Voila, x1 and x2 are our unique numbers. Reason: both the above sets contain one of the unique elements and rest elements of the sets occur twice which will get nullified after ⊕ operation.

Sample Test Case:

Input:

2 4 6 7 4 5 6 2

Output: 5 7

Apmi Kaksha

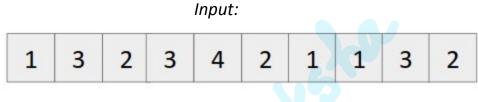
Code

```
#include<iostream>
using namespace std;
int setBit(int n, int pos) {
    return ((n & (1 << pos)) != 0);
void unique(int arr[], int n) {
   int xorsum = 0;
        xorsum = xorsum ^ arr[i];
   int tempxor = xorsum;
    int setbit = 0;
   int pos = 0;
   while (setbit != 1) {
       setbit = xorsum & 1;
       pos++;
       xorsum = xorsum >> 1;
    int newxor = 0;
        if (setBit(arr[i], pos - 1)) {
           newxor = newxor ^ arr[i];
    cout << newxor << endl;</pre>
    cout << (tempxor ^ newxor) << endl;</pre>
int main() {
   unique(arr, 8);
```

Challenge 3

Q3. Write a program to find a unique number in an array where all numbers except one, are present thrice.

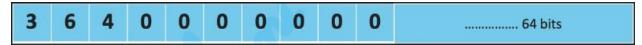
Sample Test Case:



Output: 4

Logic

1. We will maintain an array of 64 size which will store the number of times ith bit has occurred in the array.



- 2. Take the modulo of each element of this array with 3. Resultant array will represent the binary representation of the unique number.
- 3. Convert that binary number to decimal number and output it.

Apni Kaksha

Code

```
#include<iostream>
using namespace std;
bool getBit(int n, int pos) {
    return ((n & (1 << pos)) != 0);
int setBit( int n, int pos) {
    return (n | (1 << pos));
int unique(int arr[], int n) {
    int result = 0;
    for (int i = 0; i < 64; i++) {
            if (getBit(arr[j], i)) {
                sum++;
        if (sum % 3 != 0) {
            result = setBit(result, i);
    return result;
int main() {
    cout << unique(arr, 10) << endl;</pre>
```