Bit Manipulation

Count the no of set bits :

(1)

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

Time complexity: O(total bits in n)

(2) Brian - Kerningham Algorithm

Time complexity = O(No of set bits)

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

At step,

At every s...
I am turning off the last set bit.   }
return res; }.

$$\begin{array}{r} 10\textcircled{1}000 \\ 100111 \\ \hline 100000 \end{array}$$

→ Whenever one does a "n-1", then all the bits which are zero after the LSB waal first bit set includin that one converted to zero.

③ Lookup Table Based Approach

$\Theta(1)$ solution, but extra space required.

```
int table [256];
void initialize ()
 { table [0] = 0;
   for ( int i = 1; i < 256; i++)          || 0 to 255 done.
       table [i] = (i & 1) + table [i/2]
 }

int count (int n){
  int res = table[n & 0xff];
    >>8;              [n & 0xff];
n = n>>8;
res = res + table[n & 0xff];
  n = n>>8;
res += table[n & 0xff];
  n = n>>8;
  res += table[n & 0xff];
  return res;
  }.
```

→ Given an array of n numbers that has values in the range [1...n+1]. Every number appears exactly once. Hence one number is missing. Find the missing number.

You do XOR of all the numbers in the array and then XOR it with (XOR of all the numbers in the range (1...n+1)), you will get the missing number or the number that is single and every other number is appearing two times.

Finding two odd Appearing Numbers

→ Odd occurring elements are always 2.

```
void oddoccurring (int arr[ ], int n){
{      int XOR, res1, res2 = 0;
       for (int i = 0; i<n; i++)
            XOR = XOR ^ arr[i];
       int right_set bit = XOR & ~(XOR-1
                       // Rightmost set
                          bit.

       for (int i = 0; i<n; i++)
                            set bit) != 0
```

XOR of all the numbers

XOR of all the nos.
↑
the 2 odd numbers.
XOR of 1st odd occurring and 2nd odd occurring

$\Theta(n)$ Solution

```
{ if((arr[i] & rightm--
        res1 = res1 ^ arr[i];
  else
        res2 = res2 ^ arr[i];
  }
  cout<< res1 << " " << res2 << endl;
}
```

Generating Power set using Bitwise Operators

All the Subsets

```
void printPowerSet(string str){
    int n = str.length();
    int powsize = pow(2,n);
    for(int counter = 0; counter < powsize; counter++)
    {
        for(int i = 0; i < n; i++){
            if(counter & (1<<i) != 0)
                print(str[i]);
        }
    }
}
```

$\Theta(2^n * n)$

counter represents the 2^n subsets.

Binary representation.

0 000 ← We use the set bits in the binary representation to find the element subset

1 001

2 010

3 011

4 100

5 101.

)

6

⋮

2^n.

110

of the

→ The bitwise operations are found to be much faster and are some times used to improve the efficiency of a program.

→ The left shift and right shift operators cannot be used with negative numbers.

→ The & operator can be used to quickly check if a no is odd or even. The value of expression (x & 1) would be non-zero only if x is odd, otherwise the value would be zero.

Bit Manipulation Tricks

1) How to set a bit in the number 'num' ?

$$temp = (1 << n) \rightarrow \text{equivalent to } 2^n.$$

$$res = (num \underset{\text{OR}}{\underset{\uparrow}{|}} temp)$$

'num' ?

num .

2) Unsetting a bit in the number

```
temp1 = 1 << n
temp = ~temp1
res = (num & temp)
```

3) Toggling a bit at nth position:

```
temp = 1<<n
res = (num ^ temp)
```

4) Divide by 2:

```
x = x>>1;
```

5) Multiplication by 2:

```
x = x<<1;
```

6) Find log base 2 of a 32 bit integer:

```
int log2 (int x){
    int res = 0;
    while(x >>= 1)
        res++;
    return res;
}
```

7) Flipping the bits of a number:

ith all bits set in a

Value = A number with all given number.

res = value - Number.

8) Given a Number N, find the most significant set bit in the given number.

res : $2^{\log_2(N)}$

For ex:

10

↗

1010

↳ MSB

$= 2^{\log_2(10)}$

$= 8$

9) Given a Number N, the task is to find the XOR of all numbers from 1 to N.

N = 1 : 1 → 1

N = 2 : 1 ^ 2 → 3

N = 3 : 1 ^ 2 ^ 3 → 0

N = 4 : 1 ^ 2 ^ 3 ^ 4 → 4

N = 5 : 1 ^ 2 ^ 3 ^ 4 ^ 5 → 1

N = 6 : 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 → 7

N = 7 : 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 ^ 7 → 0

N = 8 : 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 ^ 7 ^ 8 → 8

N % 4

1) if rem = 0 ; then ans = N
2) if rem = 1 ; then ans = 1
3) if rem = 2 ; then ans = N+1
4) if rem = 3 ; then ans = 0

10) MAXIMUM AND VALUE

Given an array arr[] of N positive elements. The task is to find the max. AND value generated by any pair of the element from the array.

```
int checkBit( int pattern , int arr[], int n)
{
    int count = 0;
    for(int i = 0; i<n ; i++)
      { if((pattern & arr[i]) == pattern)
              count ++;
      }
      return count;
      }.

int maxAnd ( int arr[ ], int n){
    int res = 0;
     int count;
  for(int bit = 31; bit >= 0; bit --)
```

```
for (int bit = 
{
    count = checkBit (res | (1 << bit), arr, n)
     if (count >= 2)
         res = res | (1 << bit) ;

    }.
```

4	8	12	16

11) Count total set bits in all numbers from 1 to n.