Todays Content

1. Bitwise Operators
2. Interesting Properties
3. Left Shift & Right Shift
4. Problems using Left & Right Shift.

Bitwise operations : { AND, OR, XOR, Inverse, leftshift, rightshift

		AND	OR	XOR	Inv
A	B	A & B	A|B	A ^ B	~ A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Same Same puppy shame

Binary ⟷ Decimal : { internaly }

// a = 29 b = 19 :

	2^4	2^3	2^2	2^1	2^0	
a :	1	1	1	0	1	
b :	1	0	0	1	1	
print(a&b) :	1	0	0	0	1	→ = 17
print(a|b) :	1	1	1	1	1	→ = 31
print(a^b) :	0	1	1	1	0	→ = 14

// a = 20, b = 45

	2^5	2^4	2^3	2^2	2^1	2^0	
a :	0	1	0	1	0	0	
b :	1	0	1	1	0	1	
print(a&b) :	0	0	0	1	0	0	: 4
print(a|b) :	1	1	1	1	0	1	= 61
print(a^b) :	1	1	1	0	0	1	= 57

Even Odd:

Predict if below binary numbers are even or odd

Ex1:

$$\begin{array}{cccccc} & \text{even} & & 2^2 & 2^1 & 2^0 \\ 1 & 0 & \ldots & 1 & 1 & 0 \end{array} = \text{even}$$

$$\begin{array}{cccccc} & \text{even} & & 2^2 & 2^1 & 2^0 \\ 1 & 1 & \ldots & 0 & 1 & 1 \end{array} = \text{even} + 1 = \text{odd}$$

$$\begin{array}{cccccc} & \text{even} & & 2^2 & 2^1 & 2^0 \\ 1 & 0 & \ldots & 1 & 1 & 1 \end{array} = \text{even} + 1 = \text{odd}$$

#Con:

0^{th} Bit → 0 Even
0^{th} Bit → 1 Odd

Bitwise Properties

$$\begin{array}{lcccc} & 2^3 & 2^2 & 2^1 & 2^0 \\ 1.\ A = 10 : & 1 & 0 & 1 & 0 \\ 1 : & 0 & 0 & 0 & 1 \\ \hline A \& 1 : & 0 & 0 & 0 & 0 \end{array} = 0 \qquad \text{val}$$

$$\begin{array}{lcccc} & 2^3 & 2^2 & 2^1 & 2^0 \\ 1.\ A = 14 : & 1 & 1 & 1 & 0 \\ 1 : & 0 & 0 & 0 & 1 \\ \hline A \& 1 : & 0 & 0 & 0 & 0 \end{array} = 0 \qquad \text{val}$$

$$\begin{array}{lcccc} & 2^3 & 2^2 & 2^1 & 2^0 \\ 1.\ A = 11 : & 1 & 0 & 1 & 1 \\ 1 : & 0 & 0 & 0 & 1 \\ \hline A \& 1 : & 0 & 0 & 0 & 1 \end{array} = 1 \qquad \text{val}$$

$$\begin{array}{lcccc} & 2^3 & 2^2 & 2^1 & 2^0 \\ 1.\ A = 13 : & 1 & 1 & 0 & 1 \\ 1 : & 0 & 0 & 0 & 1 \\ \hline A \& 1 : & 0 & 0 & 0 & 1 \end{array} = 1 \qquad \text{val}$$

Con: A & 1
→ 0 : if A is even
→ 1 : if A is odd

Con: A & 1 → Gives Info On 0^{th} bit
→ 0 : if 0^{th} in A = 0
→ 1 : if 0^{th} in A = 1

Few More Interesting Properties

1. A&1 → 1 : odd 0 : even

2. A&0 →

A :	1 0 1 1 :
0 :	0 0 0 0 :
A&0 :	0 0 0 0 : 0

3. A&A →

A :	1 0 1 1 :
A :	1 0 1 1 : val
A&A :	1 0 1 1 = A

3. A|A →

A :	1 0 1 1 :
A :	1 0 1 1 :
A|A :	1 0 1 1 = A

2. A|0 →

A :	1 0 1 1 :
0 :	0 0 0 0 :
A|0 :	1 0 1 1 = A

3. A^A →

A :	1 0 1 1 :
A :	1 0 1 1 :
A^A :	0 0 0 0 = 0

4. A^0 →

A :	1 0 1 1
0 :	0 0 0 0
A^0 :	1 0 1 1 = A

4. A|1 : TODO

if A is Even:

	....	—	2^2	2^1	2^0
A :	1 0 1 0		1	1	0
1	0 0 0 0		0	0	1
A|1 :	_ _ _ _		_	_	_ =

if A is odd

	....	—	2^2	2^1	2^0
A :	1 0 1 0		1	1	1
1	0 0 0 0		0	0	1
A|1 :	_ _ _ _		_	_	_ =

5. A^1 : TODO

if A is Even

	....	—	2^2	2^1	2^0
A :	1 0 1 0		1	1	0
1	0 0 0 0		0	0	1
A^1 :	_ _ _ _		_	_	_ =

if A is odd

	....	—	2^2	2^1	2^0
A :	1 0 1 0		1	1	1
1	0 0 0 0		0	0	1
A^1 :	_ _ _ _		_	_	_ =

Few more properties :

Commutative:

$A \& B = B \& A$

$A \mid B = B \mid A$

$A \wedge B = B \wedge A$

Associative

$A \& B \& C = B \& C \& A = B \& A \& C$

$A \wedge B \wedge C = B \wedge C \wedge A = B \wedge A \wedge C$

$A \mid B \mid C = B \mid C \mid A = B \mid A \mid C$

XOR Calculating

1. $a \wedge b \wedge a \wedge d \wedge b = \cancel{a \wedge a} \wedge \cancel{b \wedge b} \wedge d = d$

   0   b   0   d

2. $1 \wedge 3 \wedge 5 \wedge 3 \wedge 2 \wedge 1 \wedge 5 = 2$

```
 1 : 0 0 1
^3 : 0 1 1
     -----
     0 1 0
 5 : 1 0 1
     -----
     1 1 1
 3   0 1 1
     -----
     1 0 0
 2   0 1 0
     -----
     1 1 0
 1   0 0 1
     -----
     1 1 1
 5   1 0 1
     -----
     0 1 0  = 2
```

Q Given ar[N] every ele repeats twice except 1, return unique element

Ex1: ar[] = { 1 3 5 3 2 1 5 } ans = 2

Ex2: ar[] = { 7 6 7 9 9 } ans = 6

Idea: Calculate xor of all elements & return final value

```
int unique(vector<int> &ar){   TC: O(N)   SC: O(1)
    int sum=0, N=ar.size();
    for(int i=0; i<N; i++){
        sum = sum ^ ar[i];
    }

    return sum;
}
```

left Shift: << It will move bit's to left side.

Ex: Say a is 8 bit number

MSB:		-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Decimal
a = 10 :		0	0	0	0	1	0	1	0	$= 2^3 + 2^1 = 10 = 2^0 * 10$
a<<1 :	*	0	0	0	1	0	1	0	0	$= 2^4 + 2^2 = 20 = 2^1 * 10$
a<<2 :	*	0	0	1	0	1	0	0	0	$= 2^5 + 2^3 = 40 = 2^2 * 10$
a<<3 :	*	0	1	0	1	0	0	0	0	$= 2^6 + 2^4 = 80 = 2^3 * 10$
a<<4 :	*	1	0	1	0	0	0	0	0	$= -2^7 + 2^5 = -96 = 2^4 * 10 = 160$ * ?
a<<5 :		_	_	_	_	_	_	_	_	= 160 > 8 bit range overflow.

#obs: #Note: Data can overflow

$\boxed{a << n : 2^n * a}$ # If there is no overflow.

$5 << 3 : 2^3 * 5 = 40$

$1 << n : 2^n * 1$

$1 << 3 : 2^3 = 8$

Right Shift >> It will move bit's to right Side

#Note: In rightshift MSB bit value is retained.

Ex: Say a is 8 bit number

	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
a = 20 :	0	0	0	1	0	1	0	0		$= 2^4 + 2^2 = 20$
a>>1 :	0	0	0	0	1	0	1	0	*	$= 2^3 + 2^1 = 10 = 20/2^1$
a>>2 :	0	0	0	0	0	1	0	1	*	$= 2^2 + 2^0 = 5 = 20/2^2$
a>>3 :	0	0	0	0	0	0	1	0	*	$= 2^1 = 2 = 20/2^3$
a>>4 :	0	0	0	0	0	0	0	1	*	$= 2^0 = 1 = 20/2^4$
a>>5 :	0	0	0	0	0	0	0	0	*	$= 0 = 0 = 20/2^5$

#obs:

$a >> n = a/2^n$

>> on negative Numbers

	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Decimal
a = -20 :	1	1	1	0	1	1	0	0	: -20
a >> 1 :	1	1	1	1	0	1	1	0	: $-2^7 + \underline{2^6 + 2^5 + 2^4 + 2^2 + 2^1} = -128 + 118 = \underline{-10}$

	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Decimal
a = -1 :	1	1	1	1	1	1	1	1	: -1
a >> 1 :	1	1	1	1	1	1	1	1	: -1

-1 >> 1 = -1;

Power of leftshift: Set : 1 Unset : 0

bit pos : 0 1 2 .. # Index in script

	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1 :	0	0	0	0	0	0	0	1	
1<<3 :	0	0	0	0	1	0	0	0	: 2^3

	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1 :	0	0	0	0	0	0	0	1	
1<<4 :	0	0	0	1	0	0	0	0	= 2^4

Set i^{th} Bit in N:

if i^{th} Bit in N: already set leave it
if i^{th} Bit in N: Not set, set it.
Hint: 1|1 = 1   0|1 = 1, Need to perform |

Ex 1:

N = 41 i = 2

	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
N = 41	0	0	1	0	1	0	0	1
(1<<2)	0	0	0	0	0	1	0	0
N|(1<<2)	0	0	1	0	1	1	0	1

N = 41 i = 4

	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
N = 41	0	0	1	0	1	0	0	1
(1<<4)	0	0	0	1	0	0	0	0
N|(1<<4)	0	0	1	1	1	0	0	1

N = 41 i = 3

	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
N = 41	0	0	1	0	1	0	0	1
(1<<3)	0	0	0	0	1	0	0	0
N|(1<<3)	0	0	1	0	1	0	0	1

```
int Set(int N, int i){
    N = N|(1<<i)
    return N;
}
```

Q: Given N & i, check if i^{th} bit in N is set = 1 or Not = 0?

Ex:

	2^5	2^4	2^3	2^2	2^1	2^0
N = 21	0	1	0	1	0	1

i = 2 return true;

	2^5	2^4	2^3	2^2	2^1	2^0
N = 37	1	0	0	1	0	1

i = 3 return false;

Note: Bit positions start from zero

Idea 1:

	2^5	2^4	2^3	2^2	2^1	2^0	i
N = 45	1	0	1	0	1	1	0
1							

: (N & 1) == 1.

	2^5	2^4	2^3	2^2	2^1	2^0	i
N = 45	1	0	1	0	1	1	1
N >> 1		1	0	1	0	1	

: (N >> 1) & 1 == 1.

	2^5	2^4	2^3	2^2	2^1	2^0	i
N = 45	1	0	1	0	1	1	2
N >> 2			1	0	1	0	

: (N >> 2) & 1 == 1.

	2^5	2^4	2^3	2^2	2^1	2^0	i
N = 45	1	0	1	0	1	1	3
N >> 3				1	0	1	

: (N >> 3) & 1 == 1.

```
boolean checkBit(int N, int i){
    if( (N>>i) & 1 == 1) {
        return true;
    }
    else {
        return false;
    }
}
```

return (N >> i) & 1 == 1

Set's Bit's In Range: TODO. Set i^{th} Bit [OR] Unset i^{th} Bit [XOR]

Given B, C create a binary number with B 1's & C 0's & return Decimal.

Note: No need to worry about overflows.

Ex1: B C 2^5 2^4 2^3 2^2 2^1 2^0

3 2 :

4 2 :

Idea:

Say B=5 C=?

2^4 2^3 2^2 2^1 2^0

ele = =

Say B=4 C=?

2^3 2^2 2^1 2^0

ele = =

obs1:

Say B 1's $\underbrace{\text{6 1's}}$

ele =

obs2:

Say B 1's & C 0's:

ele = $\underbrace{\text{6 1's}}$ Need to add C 0's at back

So perform leftshift C time

$\underbrace{\text{6 1's}}$ $\underbrace{\text{C 0's}}$

ele =

Final ans:

Given B, C;

ele =

ele =