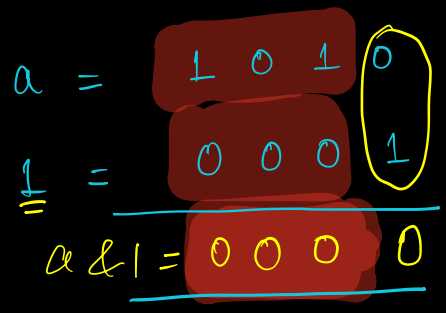
Today's Agenda :-

- → Basic bitwise properties
- → Left shift & Right Shift
- → Basic overflows
- → Bitwise Problems
  - → Check Bit
  - → Set Bit
  - → Count set bits
  - → Set x & y^{th} bits
  - → Few more problems based on time left.

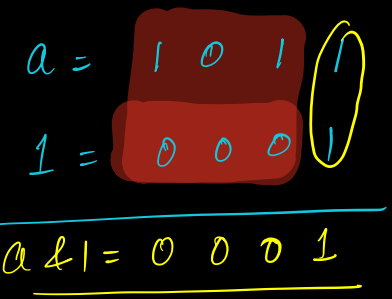
//byte a = 10 → Bitwise &

print (a & 1) ⇒ 0



//byte a = 11

print (a & 1) ⇒ 1




```
if (a & 1 == 1) {
    //a is odd
    //0th bit is a set bit in a
}
```

Very Useful

Let a is a decimal number.

a & 0 = 0	a | 0 = a	a ^ 0 = a
a & a = a	a | a = a	a ^ a = 0

Properties :-

1) $a \,\&\, b = b \,\&\, a$
2) $a \mid b = b \mid a$
3) $a \wedge b = b \wedge a$

} Commutative Property

1) $(a \,\&\, b \,\&\, c) = (c \,\&\, b \,\&\, a) = (b \,\&\, c \,\&\, a) = (a \,\&\, c \,\&\, b)$
2) $(a \mid b \mid c) = (a \mid c \mid b) = (b \mid c \mid a)$
3) $(a \wedge b \wedge c) = (a \wedge c \wedge b) = (b \wedge c \wedge a)$

} Associative Property.

Q1) $a \wedge b \wedge b \wedge c \wedge a$

$= (a \wedge a \wedge b \wedge b \wedge c) = c$

$= 0 \wedge b \wedge b \wedge c$

$= b \wedge b \wedge c$

$= 0 \wedge c$

$= c$

Q2) $c \wedge d \wedge a \wedge c \wedge a \wedge g \wedge d$

$= 0 \wedge g = g$

Q1) Given N Array elements, every element repeats even times except 1 element. Find unique element.

Ex: arr :- [3, 8, 4, 8, 3, 7, 4]

↳ ans = 7

Ex:- arr :- [2, 9, 7, 2, 7]

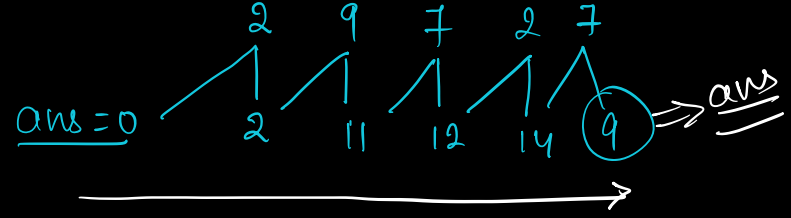
0 ^ 9 = 9

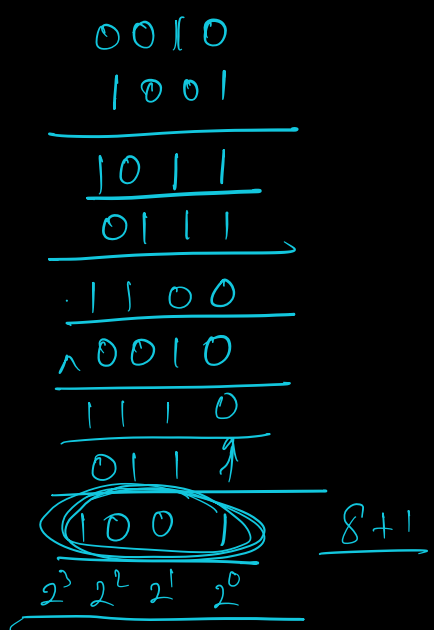
↳ ans = 9

// Idea :- get xor of all elements.

$a \wedge 0 = a$

```
ans = 0
for(i=0; i<N; i++){
    ans = ans ^ arr[i];
}
return ans;
```





// Left Shift (<<)

	8	7	6	5	4	3	2	1	0	
	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
byte a = 10 ⇒	0	0	0	0	1	0	1	0	⇒	10
a << 1 :	0	0	0	1	0	1	0	0	⇒	20 ⇒ 10 × 2^1
a << 2 :	0	0	1	0	1	0	0	0	⇒	40 ⇒ 10 × 2^2
a << 3 :	0	1	0	1	0	0	0	0	⇒	80 ⇒ 10 × 2^3
a << 4 :	1	0	1	0	0	0	0	0	⇒	-96

→ [-128, 127]

126

Ideally, this should have been 160, but 160 cannot be stored in 8 bits, hence overflow.

$$a << N = a * 2^N$$ ← Use this.

Put a = 1

$$1 << N = 2^N$$

$$a << 1 = a * 2^1$$

$$a << 2 = a * 2^2$$

$$a << 3 = a * 2^3$$

⋮

$$a << N = a * 2^N$$

↳ Assuming No overflow

// Calculate 2^N {

→ Power function x

→ Loop *

return 1 << N ;

}

00 000001 = 1

000001 0 ⇒ 2 1 << 1 ⇒ 2^1

00000100 ⇒ 4 1 << 2 ⇒ 2^2

8

16

32

int → $[-2^{31}, 2^{31}-1]$

long → $[-2^{63}, 2^{63}-1]$.

Input :- N = 29 → 2^{29}

N = 30 → 2^{30}

N = 31 → 2^{31} → overflow.

N ⇒ [0, 30]

→ long.

return (1L << N)

N = 31 → 2^{31}

N = 60 → 2^{60}

N = 63 → 2^{63}

N ⇒ [0, 62]

If N > 62:

Python : { No Issue } Implemented by strings.

Java : { Big Integer } Lots of libraries / function.

C++ : { }

C : { }

// All bitwise operators will only work with Integers.

// Suppose 5^N ⇒ $5 << N$ → Use power functions.

$a << N = a * 2^N$

$5 << N = 5 * 2^N = 5^N$

$1 << N = 1 * 2^N$

$2^N = 1 << N$

// Right Shift

$a >> 1 \Rightarrow \frac{a}{2}$

byte a = 50 : 0 0 1 1 0 0 1 0 — 50

a >> 1 : 0 0 0 1 1 0 0 1 ⇒ 16+8+1 = 25 ⇒ $50/2 \Rightarrow 50/2^1$

a >> 2 : 0 0 0 0 1 1 0 0 ⇒ 8+4 = 12 ⇒ $50/4 \Rightarrow 50/2^2$

a >> 3 : 0 0 0 0 0 1 1 0 ⇒ 4+2 ⇒ 6 ⇒ $50/8 \Rightarrow 50/2^3$

a >> 4 : 3 — $50/2^4$

a >> 5 : 1 — $50/2^5$

a >> 6 : 0 — $50/2^6$

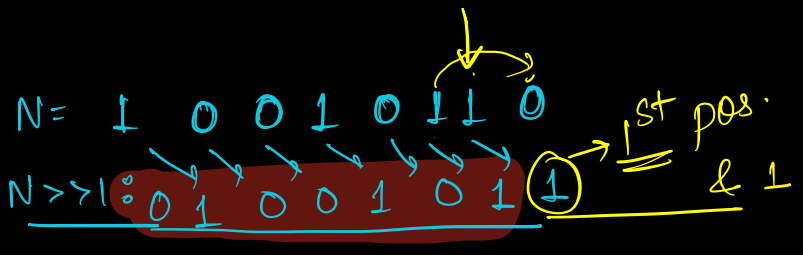
remain 0

$$a >> N = \frac{a}{2^N}$$

Q3) Given N and i, check if ith bit pos in N is set or not?

$0 <= N <= 10^9$

$0 <= i <= 30$



Ex N = 26 : 4 3 2 1 0 / 1 1 0 1 0

i = 2 return false.

Ex N = 35 : 5 4 3 2 1 0 / 1 0 0 0 1 1

i = 1 return true:

i=0 ? N & 1 == 1 ✓

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

i=2 (N >> 2) & 1 == 1

i=3 ? (N >> 3) & 1 == 1

```
bool checkBit(N, i){
    return (N >> i) & 1 == 1;
}
```

ith bit is coming at 0th position

TC: O(1)

Try using.
TODO: Left shift

Q4) Given N & i, set i^{th} bit in N.

$0 <= N <= 10^9$

$0 <= i <= 30$

$N = 26$: 4 3 2 1 0 → 1 1 0 1 0 ⇒ $26 + 2^2 = 30$

$i = 2$: 1 1 1 1 0 ⇒ 26 + 4 ⇒ 30.

2^2

$N = 35$: 5 4 3 2 1 0 → 1 0 0 0 1 1

$i = 1$: 1 0 0 0 1 1

$i = 3$: 2^3 → 1 0 1 0 1 1 → $35 + 2^3 \Rightarrow 35 + 8 \Rightarrow 43$

$N =$ 7 6 5 4 3 2 1 0 → 1 0 0 1 0 0 1 1 0

$i = 4 \Rightarrow$ 0 0 0 1 0 0 0 0

1 0 0 1 0 1 1 0

1	1	1 = 1
0	1	1 = 1

0	1	1 = 1
1	1	1 = 1

i^{th} bit

2^i ⇒ only i^{th} bit is set

$N \mid 2^i$

$N =$ 7 6 5 4 3 2 1 0 → 1 0 0 1 0 1 1 0

$i = 5$: 0 0 1 0 0 0 0 0

1 0 1 1 0 1 1 0 → set i^{th} bit

```
set (N, i) {
    if (! checkbit (N, i)) {
        // if ith bit is not set
        // we need to set ith bit
        N = N + (1 << i) ; // adding the contribution of ith bit => 2^i
    }
    return N;
}
```

```
return N | (1<<i);
```

N: ith set | ith bit set

```
1 | 1 = 1
```

N: ith unset | ith bit set

```
0 | 1 = 1
```

		0	
	1	1	2^0
	1	0	2^1
2			
1	0	0	2^2
		2^i	

Q5) Set x^{th} & y^{th} bit in Number 0.

$0 <= x <= 30$

$0 <= y <= 30$

Ex $x=2, y=5$

7	6	5	4	3	2	1	0	
0	0	1	0	0	0	1	0	} = 36

$2^3 = 01000$

$2^4 = 10000$

11000

Ex $x=3, y=4$

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	0	0	} = 24

```
if(x != y)
  (1<<x)+(1<<y)
else
  (1<<x)
```

Ex $x=2, y=2$.

0	0	0	0	0	1	0	0	⇒ } 4
0	0	0	0	0	1	0	0	
0	0	0	0	0	1	0	0	

$a \mid a = a$

return $(1 << x) \mid (1 << y)$; ✓

Q6) Given N, Calculate No of set bits are there in N.

$0 <= N <= 10^9$ (int)

{ at how many bits = 1 }

Ex: N = 150 → 1 0 0 1 0 1 1 0 (7 6 5 4 3 2 1 0) ⇒ ans = 4

Ex: N = 35 → 1 0 0 0 1 1 ⇒ ans = 3.

32 bits : [0 - 31]

Approach:

```
int countBits(N){
    cnt = 0
    for(i=0; i<32; i++){        → your own checkbit.
        if(checkBit(N, i)){
            cnt++;
        }
    }
    return cnt;
}
```

	7	6	5	4	3	2	1	0	cnt
N:	1	0	0	1	0	1	1	0	0

(N & 1 == 1)

N >> 1: 0 1 0 0 1 0 1 1 1

N >> 1 ← 0 1 0 0 1 0 1 2

N >> 3 3

4

countBits(N) {

copy N

c = 0

Approach 2

```
while(N > 0){
    if(N & 1 == 1){
        c++;
    }
    N = N >> 1;
}
```

⇒ whether 0th bit inside is set or not

} ⇒ Bringing everything on right by 1.

}

Q7) Given N, check if it's a power of 2 ?

Ex 1 N = 2 → 10 ✓

Ex 2 N = 16 → 10000 ✓

Ex 3 N = 20 → 10100 X

Ex 4 N = 11 → 1011 X

Ex N = 22 → 10110 X → 100 ✓

Ex N = 4

TODO :→ { No Loop, No Predefined Function, Single line } Try