Bitwise operators + Number Systems

- When we write int a = 10 ; (in any language) internally whatever the things that's you're doing. all the servers that're running, etc. So internally it's all a bunch of 0s1.
- Computers understands only binary language (0 & 1)
- We don't code in 0 & 1 because it will become very complex. So, that's why we have programming languages.
- So, that's how numbers are stored internally this is known as no system.
- There are various no. systems.
  - base 10, base 8, base 16, base 2

Q. What "base" represents?

Ans: Base basically means How many numbers do we have in the particular base concept.

* We have been studying since we're kids: decimal no system (base 10).

- Base 2 : It means binary no. system. This is what the computer understands.

Now we can imagine 0 as false And 1 as True

wise.

Bit operators :-

1) And :-

(In words) :- Here let's say if you've 2 no, all of them should be true.

Truth-table :-

a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1 - (all are true)

Conditions :-

AND conditions means that all the numbers or input should be true, then only the entire expression will be true.

If one of the input is flase then the output will be false only.

If we're by saying of output of A & B & C then all the inputs should be true or equal to one (1)

In computer science we can say that 1 represents True and 0 represents False.

electronic :-

If 1 & 1 all the gate will be open so there will be flow of current, if 1&0 or any thing other it won't. AND gate is a basic logic gate.

Note:
(i) When you AND 1 with any number, the digits will never remains the same.

(ii) Eg:
```
  1 1 0 0 1   - same
& 1 1 1 1 1
-----------
  1 1 0 0 1   - same
```

3) OR:
in words :- It is opposite of AND gate. If any one of the input is true then the entire expression is true.

Truth-table :-

a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

Condition:-
Even if both the inputs are true output will be always positive

3) XOR (^) :- (If and only if)
In words :- It is exclusive OR. Here if you've two no. only one of them should be true and anything else will false as a output.

Truth-tables

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Condition:- If more than 2 no then, the odd no should be true.

Observation :-

1) $a \wedge 1 = \bar{a}$

Note :- $\bar{a}$:- it is basically a complementry bar that means of opposite of that no.

Eg :- ① $0 \wedge 1 = 1$

② $1 \wedge 1 = 0$

So, anything we X-OR with '1', you will get the answer opposite of that.

2) $a \wedge 0 = a$

3) $\bar{a} \wedge a = 0$

Eg :- $32 \wedge 32 = 0$

If XOR the same input the output will always be zero.

1) Complement ($\sim$) :-

Suppose :- $a = 10110$

$\bar{a} = 01001$

A	B	C	D	o/p	
0	0	0	0	0	
0	0	0	1	1	
0	0	1	0	2	
0	0	1	1	3	
0	1	0	0	4	
0	1	0	1	5	
0	1	1	0	6	
0	1	1	1	7	
1	0	0	0	8	
1	0	0	1	9	
1	0	1	0	10 →	A
1	0	1	1	11 →	B
1	1	0	0	12 →	C
1	1	0	1	13 →	D
1	1	1	0	14 →	E
1	1	1	1	15 →	F

** Nubnber system :-

1) Decimal :- 0 to 9 & it's of base 10.

Base 10 means there are 10 nos. which we can use to represent any ~~nub~~ no. in decimal form.

Eg:- $(357)_{10}$, $(10)_{10}$

2) Binary :- 0 & 1 & it's of base 2.

Eg:- ① $(10)_{10} \rightarrow (1010)_2$

② $(7)_{10} \rightarrow (0111)_2$

3) Octal → 0 to 7 base 8

Now comparing Decimal & octal. In octal we cannot 8 & 9 because we can only use the nos of from 0 to 7.

Decimal :- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
10, 11, 12 ...

Octal :- 0, 1, 2, 3, 4, 5, 6, 7, 10,
11, 12, 13, 14, 15, 16, 17, 20 ...

So, 8 in decimal is 10 in octal & 9 in decimal is 11 in octal.

4) Hexadecimal :- 0-9 & A-F base 16

Eg :- $(10)_{10} = (A)_{16}$
$(12)_{10} = (C)_{16}$

* Conversions :-

Note:

1) Decimal ~~tens~~ to base b

Q Convert $(17)_{10}$ to base 2

pt : Q keep dividing by base. take remainders, write in opposite.

2	17	
2	8	1
2	4	0
2	2	0
	1	0

$(1001)_2$

$\therefore (17)_{10} : (1001)_2$

So, when you write int a = 17 computer will store it like (1001).
Each ram has their individual cells.
So in ram it will be stored something like

a → [1 | 0 | 0 | 1] & a will be pointing towards it the cell.

Q $(17)_{10} : (?)_8$

8	17
	2 1

$\therefore (21)_8$

$\therefore (17)_{10} : (21)_8$

② Base b to decimal

① $(10001)_2 \rightarrow (?)_{10}$

Steps:

① Multiply & add the power of base with digits.

1	0	0	0	1
2^4	2^3	2^2	2^1	2^0
$2^4 \times 1$	$2^3 \times 0$	$2^2 \times 0$	$2^1 \times 0$	$2^0 \times 1$

$= 16 + 0 + 0 + 0 + 1$

$= 17.$

$\therefore (10001)_2 = (17)_{10}.$

② $(21)_8 = (?)_{10}$

2	1
8^1	8^0
$8^1 \times 2$	$+ 8^0 \times 1$

$= 16 + 1$

$= 17$

$\therefore (21)_8 = (17)_{10}$

If you want to double the no.
Just << by 1.

continuing with operators:-

⑤ Left shift operator (<<) :- It shifts all the bits towards the left side. (one by one).

Eg:- $(10)_{10} = (1010)_2$.

a 10 << 1 - It basically means that shift i/p by 1 ;
i/p

Step I: Convert it internally it into binary no.

∴ 1010 << 1

Step II Okay. Shift it towards the left. (one by one.

∴ 1 0 1 0 << 1 = 1 0 1 0 0

Step III After shift the to the left is we will require one extra no. So whenever we need an extran no in left shift operator we add 0. as above method.

∴ 1 0 1 0 0

$= 2^4 + 2^3 + 2^2 + 2^1 + 2^0$

$= 16 \times 1 + 0 + 4 \times 1 + 0 + 0$

$= 16 + 4$

$= 20$

Hence, any no. "a << 1 = 2a" means it's going to double that no.

General point :- If you left shift a no. "a" "b" times it's going to multiply $a \times 2^b$.

= $a << b = 2b$ I $a \times 2^b$

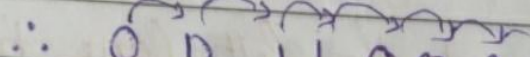
⑥ Right shift operator :- " >> "

It is opposite of left shift operator.

It moves the given i/p towards the right. Consider

① 00110011 >> 1

This question basically says that move the given i/p by 1, and by moving them one by one towards the right. we've to discard the 1.



∴ 0 0 1 1 0 0 1 ⊗ >> 1

= 00 1 1 0 0

Note:

- Similarly like in decimal no. system when the no like $(000123)_{10} = (123)_{10}$.
- Here the leading zeros are ignored
- This is same for all of the no. system. Even though because the zero from left hand side will be ignore, so the values will remained unchanged but we can't do that from right side as it will change the whole value.

∴ = 1100

Note:

In binary 1+1 = 10 & 1 is carry so ans = 0.

Eg :- $\begin{array}{r} 10 \\ +\ 11 \\ \hline 100 \end{array}$

Note: In the right shift operator we're dividing the no by 2.

$$a >> b = \frac{a}{2^b} \quad \text{- (general point).}$$

//

* Working :-

a. Given a no n find if it is odd or even.

Note:

① When you AND a 1 with any no. the digits remain the same.

② We know that any no, whatever calculation we do internally it will be calculated as a binary no. even if you do substraction, addition, etc.

Eg

$12 + 7 \rightarrow$

$$\begin{array}{r} 1100 \\ +0111 \\ \hline 10000 \end{array} \qquad \begin{array}{r} 1100 \\ +0111 \\ \hline 10011 \end{array}$$

$(19)_{10} \rightarrow (10011)_2$

$$\begin{array}{ccccc} 1 & 0 & 0 & 1 & 1 \\ 2^4 & + 2^3 & + 2^2 & + 2^1 & + 2^0 \end{array}$$

$$2^4 \times 1 + 0 + 0 + 2 \times 1 + 1 \times 1$$

$$= 16 + 2 + 1$$

$$= 19 \text{ //}$$

Note :-

Every no. Eg :- 1001(1) leaving this, every other is a power of 2.

Hence this is the no. whether it determ because we know that leave the last one ~~ever~~ entire no. will be even (always) Why? Because all of those these are power of 2.

the last no will always be a^0. and anything positive no. raise to 0 is 1.

Hence, if this particular no is 1 it means that the answer to everything is +1.

if 2^0 place == 1 → e no. is odd.
otherwise no. is ~~evel~~ even.

find out what last digit is. if it is 1 then the no. is odd & if it is 0 then the no is even.

Q And the no & get the last digit.

```
  100101
And'ed 000001
--------
  000001
```

(1)

Hence, odd no.

Sum up:-

n & 1 == 1 → odd else even

Code

```
public Clas EvenOdd {
    public static void main (String [] args){
        int n = 67;
        System.out.println (isodd(n));
    }
    private static boolean isodd (int n){
        return (n & 1) == 1;
    }
}
```

Note:-

- Eg :- 0 1 1 0 0 0 [1] → it is known as LSB (Least Significant Bit).

1. ~~Your~~: you're given an array of numbers and in that array every number appears twice only one number appears once. you're to find that no.

arr = [2, 3, 4, 1, 2, 1, 3, 6, 4]
→ ans

So How'll you can find His?

Note:- Bit wise operators like in normal maths operator they also follows the associative properties

① eg :- (5 * 3) * (5 * 4) = 5 * 5 * 3 * 4 or 5 * 4 * 3 * 5, etc

So the order basically does not matter

② 2 ^ 3 ^ 3 ^ 4 = 2 ^ 4 ^ 3 ^ 3, etc.

We wanna do it in O(1) & in one single pass

Note:
As we know any no ^ with the same no.
a ^ a = 0 & 0 ^ a = a

So, if ^ of the entire array the ans. I know the all the duplicates will lead to zero

Ans: XOR all the numbers

time complexity: O(n)
space complexity: O(1)

2Q. arr = [-2, 3, 2, 4, -5, 5, -4]
ans: 8

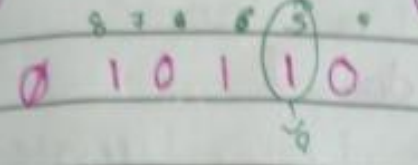
Note: ORDER does not matters

#Code:

Ans
```
public class Unique {
    public static void main(String[] args){
        int[] arr = {2, 3, -3, -2, 6, 5, -5};
        System.out.println(unique(arr));
    }

    private static int unique(int[] arr){
        int ans = 0;
        for(int i : arr){
            ans ^= i;
        }
        return ans;
    }
}
```

Q. find i^{th} bit of the number?



Ans: So we know that we were able to find How do find LSB or right most bit we just And it & 1
So And the particular digit with 1, it will give 1 no matter what no. rest And will be 0.

∴ Ans:-

$$\begin{array}{r} 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0 \\ \&\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0 \quad \text{- Mask} \\ \hline 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0 \end{array}$$

Q. How do get 00010000?

Note

Mask is a sp. seperate no. or entity you've that allows us to get our answer related to it.

n = mask with n-1 zeros. Q. How do we do that?
So we need n-1 zeros in the right hand side
Q What do we need to do in order to move the 1 towards the left hand side get zeros over there
Left shift (1 << (n-1))
∴ 1 << 4 = 10000

Ans: n & (1 << (n-1))

Q. Set the i^{th} bit

Set means: turn it to 1

so if i^{th} bit 0 make it 1
if i^{th} bit 1 remain 1 only.

1 0 1 0 1 1 0

let's say you want to set the 4th bit.

Note:

0 ^ 1 : 0 OR 1 = 1

1 0 1 0 1 1 0
OR 0 0 0 1 0 0 0 - {Mask}
1 0 1 1 1 1 0

Q. Reset i^{th} bit

1 0 1 0 1 1 0

Reset means: if it's 1 make it 0
if its 0 remain 0.

Note: If we And any no with 1 we'll get the no itself but if we And any no. with 0 we'll get 0 only
0 & 1 = 0

1 0 1 0 1 1 0
& 1 0 1 0 1 1 1 - {Mask}
1 0 1 0 1 1 0

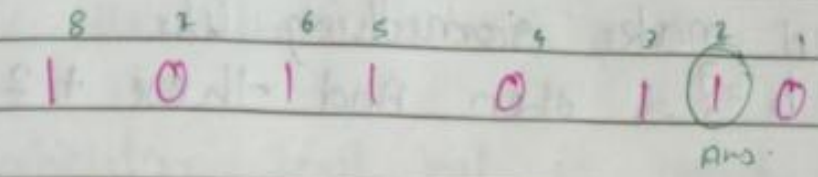
Q So how did get that Mask?

Ans By complement

$$Mask = \overline{(1 << (n-1))}$$

Q Find the position of the right most set bit?

Eg:



Looking from the right hand side which is the set bit?
The first 1 that occurs from the right hand side.

∴ Ans = 2

1 0 1 1 0 1 1 0

↓

If we're writing this no. in the form of if we do this one right most bit.

n = a 1 b

Or

1 0 1 1 0 1 1 0 0

a b Ans = 4

n = a 1 b

a = 1 0 1 1 0 1

b = 0 0

$1011011\ (1)\ 00$

a b

As we know 100 is our answer so we're not touch touching them.
And we have to create 101101 all of them as 0
we need make something like
$\bar{a}\ 1\ b = ?$ & then And these + 2.

in terms of formula it's equal to $-n$

So, let's say

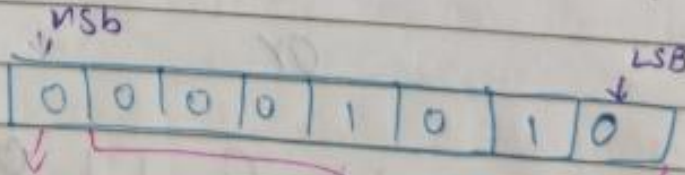
$-n = \bar{a}\ 1\ b$

Ans : $n \& (-n)$ $\because a \& \bar{a} = 0$ //

* Negative of a number

1 byte = 8 bits
So we can store 8 nos, 8 time 0 or (0, 1) in one byte.

Eg: 10 :



-10 = ?

Note:-

① MSB : Most Significant Bit [If no. is positive or neg
1 = neg & 0 = positive

② LSB : Least Significant Bit. (If no is even or odd)

Note:

- MSB is the reserved bit because if it is stored with 1 that means it is a negative no. or 0 then that means it is a positive no.

- Size of an it integer = 4 bytes. = 32 bits
So, Here the first bit is gonna be either 1 or 0.
That particular first bit is going to represent whether the no. is positive or not.
And rest of the 31 bits thats going to represents the value of the number.

- ~~Two~~ 2's Compliment

Steps

1) Take the compliment of the given input

2) Add 1 to it.

Eg: $(10)_{10} = (00001010)_2$

∴ ① 11110101

② 11110101 + 1 = 11110110

∴ Ans = $(-10)_2 = (11110110)$

This is how you calculate negative because in binary we can not apply - symbol.

Q Why does 2's complement gives negative of a number?

Ans:

- As we know when the size is 1 byte (8 bits) and you're bay having an additional bit, it that's going to get ignored.
- Eg: 1 0 1 0 1 1 0 1 1 1
  Here as we know that 1 byte is of 8 bit only so it will discard the starting 10 because of the space.
- As we know that we substract 0 from a number ans always gonna be in negative.
  Eg2: 1 0 0 0 0 0 0 0 0 0 → As this should give neg 10
  - 0 0 0 0 1 0 1 0

As since we're storing this neg no in the size of 1 byte if add additional on in the begining of extra 0 (1) that won't really matter because that's gonna get ignored either way.
because the size will not ~~swallow~~ allow ~~that~~ them.

- If we are only allowed to add 8 bits, 9 or more than that will get discarded as the limit is of 8 bits.
- We can see that 1 0 0 0 0 0 0 0 0 is a power of 2, if there is one single 1 is available in the binary representation no rest everything is 0 that means it is of power of 2.
- Eg: 8

∴ 8 = 1000

= 7+1 = 0111 + 1

∴ 0111

+ 0001

1000

or 16 = 15+1 = 1111 + 1

1111

+ 0001

10000

• so we can write this in entire thing as

eg: 100000000
- 00001010

as: 100000000 = 11111111 + 1

so the question reduce to

11111111 + 1 - 00001010

(rearrange) 11111111 - 00001010 + 1

complement why?

```
  11111111
- 00001010   } complement
  11110101
```

(1-0=1, 1-1=0)

No 2's complement's step 2: +1

* Range of numbers

1 byte : 8 bits : $2^8 = 256$

so total 256 no's we can make.

| 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 |

2 x 2 x 2 x 2 x 2 x 2 x 2 x 2

= 256.

Total numbers of unique no that we can store in a datatype of size 1 byte is 256.

So, actual no will be 0 to 7 bits

total bits n-1 = 7 bits.

Total no we can be make from 7 bits is

$2^7 = 128$

∴ 128 no in negative & 128 in positive
But even 0 counts & as we know neg of 0 is not
∴ -128 to 127 (range).

→ Range formula :- for n bits.

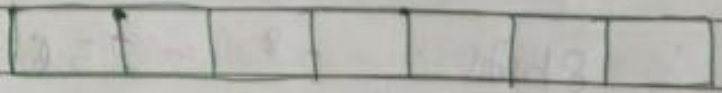
-2^{n-1} to $2^{n-1} - 1$

① Every no. is appearing 3 times & 1 no. is appearing only once : find that no.

arr = [2, 3, 2, 2, 7, 7, 8, 7, 8, 8]

idea :- we know every no is appearing 3 times than the values of its bits will also be appearing 3 times

if we take an empty array :-



2 :- 10

Basically this array says that all the bits that're set means that are 1. just keep that adding those in this set array.

so if we're adding 2 over here and then we are adding 2 again in this means that the bits available over there is it just going to added as 1.

2 :- 1 0
2 :- 1 0
2 :- 1 0
3 :- 1 1
7 :- 1 1 1
7 :- 1 1 1
8 :- 1 0 0 0
7 :- 1 1 1
8 :- 1 0 0 0
8 :- 1 0 0 0

So if we just count a total no. of bits then it'll become

1					3	3	7	4	

Imagine there was no. 3. So ↑ this will be something like | 3 | 3 | 6 | 3 |.
So obviously if 3 was not in there array it means the every single no is appearing 3 times means their set bits are also going to be appearing 3 times. So the set bits are going to be multiple of 3.
Hence the entire no is going to be divisible by 3, every digit of this no. in the array is going to be multiple of 3. one of these no.s
But here we're say is let's say add that is some extra, So some of the digits may not be multiple of 3. will be having 7 & 4 because of 1 & 1

So if we just take the modulo "%. 3" (the no which is repeating)

$\therefore$ %. 3 = 0 0 11 = 3 // ans.

an extra no. that is our answer "Everything that was appearing 3 times their set-bits will also appear 3 times Hence, the sob set-bits will be % of 3 "i.e remainder of 3 will be 0. But if there is one extra no. i.e. 1 actually addition to the 3 multiples that we have so that won't be a multiple of 3.":

Note:

Not just of for 3, we can do this for any Eg:- for 3, 5, 7 ..... do it like-wise.

Answer:-

Q. find the n^{th} magic no.

Eg:- 1) 1 = 0 0 1 = 5 (magic no.)
$5^3 + 5^2 + 5^1$

2) 2 = 0 1 0 = 25
$5^3 + 5^2 + 5^1$

3) 3 = 0 1 1 = 30
$5^3 + 5^2 + 5^1$

4) 4 = 1 0 0 = 125
$5^3 + 5^2 + 5^1$

let's say n = 6

① convert the n into binary. ∴ = 110
② We're gonna start 5⁰ and keep on multiply these with these no. & then add them.
③ Now, we have to basically check what is the last no. & we can do that with & of 1. (And of 1)

∴ n & 1 = This will give last digit in binary representation

Note:
We don't really have to convert "n" in binary. We can directly do n & 1 it will automatically converts in to binary internally.

Okay now we've 0 so leave & move to next by right shift 1
check if that 1 then proceed.

by doing n >> 1 these will go in loop

So we'll first take
n & 1 * which will give us 0 * 5⁰
n >> 1 + ...
&
∴ 0 * 5⁰ + 1 * 5² + 1 * 5³ + ~~0 * 5⁴~~

Code :-

```
pubic class MagicNumber {
    public static void main(String[] args){
        int n = 6;

        int ans = 0;
        int base = 5;

        while (n > 0) {
            int last = n & 1;
            n = n >> 1;
            ans += last * base
            base = base * 5;
        }
        System.out.println( ans);
    }
}
```

Q find no. of digits in base b.

Eg :- $(6)_{10} = 1$

$(6)_{10} = (110)_2 = 3$

formula :

No of digits in base b of no. n =
int $(\log_b n) + 1$

~~log a b~~

$$\log_b a = \frac{\log_x a}{\log_x b}$$ //

exp: $\log_b a = x$ ... (a)

$\Rightarrow a = b^x$

similarly

$\log_2 6 = x$

$6 = 2^x$ so this base thing ↑ represents the no of digits in the

Eg: $\log_2 10 = 3.32$

$10 = 2^{3.32}$

this basically means that how many times the 2 has been multiplied to form 10.

so if we take an int value of (3.32) & and 1 to it then it will gives us the no. of digits.

Q How many no. of digits are there in the binary representation of "10"?

Ans: $\log_2 10 + 1$

Code :-

```
public class NoOfDigits {
    public static void main (String[] args){

        int n = 34567;
        int b = 10;

        int ans = (int)(Math.log(n) / Math.log(b));
// if we want to convert anything to base b, just divide it by the same log of that with b.

        System.out.println(ans);

    }
}
```

meaning :-

$$\log_b a = \frac{\log_x a}{\log_x b}$$

Time complexity :- O(n)

Pascal's Triangle

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5   10  10  5   1
    1   6   15  20  15  6   1
```

find the sum of n^{th} row?

Ans:

sum of each row = sum of all the no.

$\downarrow$

$^nC_0 + \cdots$

i.e

sum of each row = $^nC_0 + {}^nC_1 + {}^nC_2 + {}^nC_3 \cdots + = 2^n$

for n^{th} row, sum = 2^{n-1}

$\therefore 1 << (n-1) = 1 \times 2^{n-1}$

Q find out the given no. if it's power of 2 or not

"Here only 1 bit will be there that will have 1 rest everything will be 0."

Eg:- ① 100000

It is a power of 2

② 100010

It is not a power of 2

Because here we have 2 1's. but only 1 '1' should be there. Rest everything should be 0.

So that's how we figure out the given no is of power of 2 or not

as we know that

$$100000 = 11111 + 1 \quad (n-1)$$

And it ⓪

$$\begin{array}{r} 100000 \\ \& \; 011111 \\ \hline 0 \end{array}$$

Eg:- ②

$$\begin{array}{r} 10010 \\ \& \; 01111 \\ \hline 00010 \end{array}$$

= not a power of 2

Ans: If $n \& (n-1) = 0$ then it is power of 2.

Code :-

```
public class PowOfTwo {
    public static void main(String[] args){
        int n = 16;
        boolean ans = (n & (n-1)) == 0;
        System.out.println(ans);
    }
}
```

Q find a^b ?

Eg :- $3^6 = 3 \times 3 \times 3 \times 3 \times 3 \times 3$ // O(b) - 1 way

$3^6 = 3^{2+4} = 3^2 \times 3^4$ - 2 way

$3^6 = 3^{110}$

$\therefore n = 110$

ans = 1

$n \& 1 \rightarrow 0$ if 0 then ignore

PTO

"If 0 then ignore because we know $3^6 = 3^{2+4}$ so we're only taking 2 & 4 the ones that are equal to the set bits".

Now it's gonna be obviously true as we know $2^0 \times 0 = 0$ so it will not do anything. Hence, we can ignore it.

& we can say that

$= ans = ans \times base$.

$\therefore ans = 9$ & $n = 1$ now $1 >> 1 = 1$

& obviously base · base $\times$ base $= 9 \times 9 = 81$ because it is doubling base everytime

because if we have

3^{110^6} = $3^4 \times 3^2 \times 3^0$

(base is doubling everytime) $= 3^4 \times 1 \times 3^2 \times 1 \times 3^0 \times 0$

Now instead of 6 times we are running it the no. of digits in 6 // $(O \log(b))$

ans = 81

$\therefore$ ans: $81 \times 9 = 729$

Note: "Checking the last value whether it is 1 or 0. if it's 1 then multiply the ans with base."

- Base = Base $\times$ Base - do this always.

even if we are skiping 3*0 that doesn't mean 3 not will get skip.

we are only not multiplying when the value is 0 of that particular index.

Code :-

```
public class Power{
    public static void main(String[] args){
        int base = 3;
        int power = 6;

        int ans = 1;

        while (power > 0){
            if ((power & 1) == 1){
                ans = ans * base;
            }

            base = base * base;
            power = power >> 1;
        }
        System.out.println(ans);
    }
}
```

$= O(\log(\text{power}))$

Q Given a no. find the no. of setbits in it

Eg: n = 9

∴ n = 1001 Ans :- 2.

n & (-n) = 0001 → right most setbit

if we subtract with this n

n - [n & (n-1)] . 1000 ——— (1) count

So keep doing this thing till n is greater than 0. "Because we are finding the right-most bit and we're deleting it one by one"

Eg: n = 1001
& 1000
1000 ——— (1) count → 8

8 & 7 = 1000
& 0111
0000 ——— (2) count.

No. of set bits = no of iterations.

O(logn)

Code :-

```
public class SetBits {
    public static void main(String[] args){
        int n = 626;
        System.out.println(Integer.toBinaryString(n));
        System.out.println(setbit(n));
    }

    private static int setbit(int n) {
        int count = 0;

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

```
while(n>0){
count++;
n -= (n & (-n));
}
```

Q find XOR of no. from 0 to a (codeforces)

Ans let's say a = 0

	a	XOR from 0 to a
(1)	0	0
(2)	1	0^1 = 1
(3)	2	0^1^2 = 3
(4)	3	0^3^3 = 0
(5)	4	0^0^4 = 4
(6)	5	4^5 = 1
(7)	6	1^6 = 7
(8)	7	7^7 = 0
(9)	8	0^8 = 8
(10)	9	8^9 = 1

PTO →

"Here we can see a pattern 0 = 0, 4 = 4, 8 = 8, etc. & 1, 5, 9 = 1"

So what is happening?

→ An every fourth (4th) no is 0. So basically if you see the above pattern you'd say After every 4 a. pattern is repeating.

① If a % 4 = 0 - 1st no in the patter.
Ans = 0 $\xrightarrow{\text{XOR}}$ a = a(0)

② If a % 4 = 1
Ans = 1

③ If a % 4 = 2
Ans = a+1 Eg:- for 2 = 2+1 = 3 & 6 = 6+1 = 7

④ If a % 34 = 03
Ans = 0

Conclusion:

This is going to keep on repeating (all 4).

So if you want ans of XOR for 0 to 9 ↑ is the Ans.

8. XOR of all no. betn a & b

Ans

a = 3 & b = 9 - suppose

then you've to calculate

3 ^ 4 ^ 5 ^ 6 ^ 7 ^ 8 ^ 9 = ?

If we ~~take~~ talk about from 0 to 9

0 ^ 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 ^ 7 ^ 8 ^ 9 - (1)

As we know that we can calculate from 0 to 9 very easily using previous cases.

But we've to find 3 till 9

So "0 ^ 1 ^ 2 ^" are the extras &

we don't want this. we only want it from 3 till 9

How do we ~~do~~ remove these extras?

These extras are already in the entire XOR. So if we XOR "0 ^ 1 ^ 2 ^" again from the eqn (1) then the 0 ^ 1 ^ 2 ^ will get removed

we basically XOR these entire thing then

0 → (a-1) Ans = f(b) ^ f(a-1)

∴ f(a) → XOR of 0 → a

So we are adding or it already added in f(b) part & we're XOR it again so this will be duplicate &

we know that x ^ x = 0

f(b) : It represents the XOR of 0 till b

Note.

If you do XOR for every single no. then it will be very bad complexity. But if you try it will TLE "Time limit exceed" error.

Code :-

```
public class RangeXor {
    public static void main(String[] args){
        int a = 3;
        int b = 9;

        int ans = xor(b) ^ xor(a-1);

        System.out.println(ans);
    }
    static int xor(int a) {

        if (a % 4 == 0) {
            return a;
        }

        if (a % 4 == 1) {
            return 1;
        }

        if (a % 4 == 2) {
            return a+1;
        }

        if (a % 4 == 3) {
            return 0;
        }
    }
```

or

```
return 0;
```

Google.

8. Flipping an image.
Given a binary matrix A, flip the image horizontally, then invert it & return the resulting image.

Ans
① Here every row is reversed.
Eg :- 1,1,0 -> 0,1,1
(reverse an array)

② Invert it 0 → 1 or 1 → 0
(It's taking a complement)

If we XOR any no. with 1. we'll get the inverse of that no.
Eg :- If no is 1 then 1^1 = 0
but if no. is 0 then 0^1 = 1
} flipped.
So 1 becomes 0
& 0 becomes 1

So after reversing every row we don't to do anything with it. So while reversing the element we can also apply XOR

Eg :-
1 1 0 → reverse → 0 1 1
1 0 1 → 1 0 1
0 0 0 → 0 0 0

↓ invert & (flipping the no)

1 0 0
0 1 0
1 1 1 // Ans.

Just reverse every single array because we know that in 2D array every single row is individually itself an array.

Code :

```
Class FlipImage{
    public int[][] flipAndInvertImage(int[][]
                                      image){

        for (int [] row : image) {

            // reversed .
            for(int i=0; i<(image[0].length +1)/2
                                      i++){

            // to Swap
                int temp = row[i] ^ 1;
                row[i] = row[image[0].length-i-
                                      ^1];

                row[image[0].length-i-1] = temp;
            }
        }
        return image;

    }
}
```