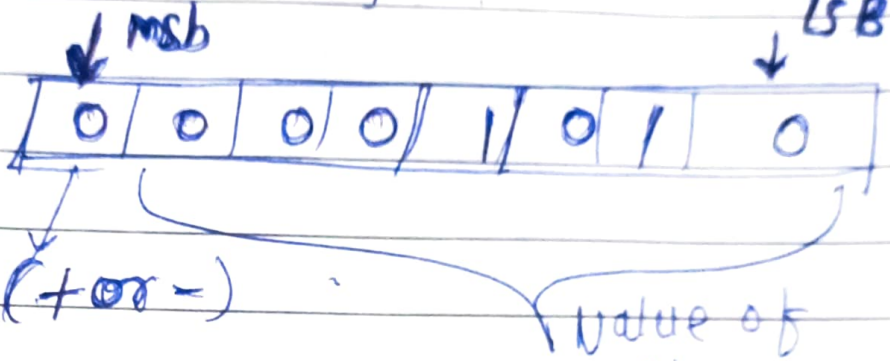
* Negative of a number

→ 1 byte = 8 bits
So we can store 8 no 8 time (0, 1) in one byte



E.g: 10

-10 = ?

Note

(1) MSB = most significant Bit
(If no is positive or negative)
Imp → (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 starts with 1 that means, it is a negative no. or 1 then that means - it is a positive no.

- Size of an int integer = 4 byte
  = 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.

- 2's Complement

Steps

1) take the compliment of the given input

2) Add 1 to it

E.g $(10)_{10} = (0000\ 1010)_2$

∴ ① 1111 0101

\+ 1

1111 0110

∴ Ans $(-10)_2 = (1111\ 0110)$

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

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

→ • As we know when the size is 1 byte (8 bits) and you're having an additional 1 size that's going to get ignored

• e.g 1010 1101 1111

Here as we know that 1 byte is of 8 bit or by so it will discard the starting of 10 because of the space..

• As we know that we subtract 0 from a number ans always gonna be in negative

e.g. :- 1 0 0 0 0 0 0 0 0

0 0 0 0 1 0 1 0

as this should give neg 10.

As since we're storing this neg no in the size of 1 byte if add addit. on in the begining of extrem o that won't really matter because that's gonna get ignored either way because the size will not allow them.

- 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 10000 0000 is a power of 2. if there is one single 1 is available in the binar representation so rest everything is 0 that means it is of power of 2
- e.g. 8

$\therefore 8 = 1000$

$\therefore 7+1 = 0111+1$

$$\begin{array}{r} 0111 \\ +\ 0001 \\ \hline 1000 \end{array}$$

or 16 same

$15+1$

$$\begin{array}{r} 1111 \\ +\ 0001 \\ \hline 10000 \end{array}$$

- So we can write this entire thing as

e.g. 100000000
$-\ 000010 10$

as :- $100000000 = 11111111 + 1$

So the question reduce to

$11111111 + 1 - 00001010$
$11111111 - 00001010 + 1$

complement thy

$$\begin{array}{r} 11111111 \\ -\ 00001010 \\ \hline 11110101 \end{array}$$] complement
(1 − 0 = 1
& 1 − 1 = 0)

No 2's Complement is 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

$2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2$

$= 256$

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 bit

total bits $n-1 = 7$ bits

total no we can 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 bit;

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

Q Every no. is appearing 3 times & 1 no. is appearing only one find that no

arr = [2, 3, 1, 2, 2, 7, 7, 8, 7, 8, 8]

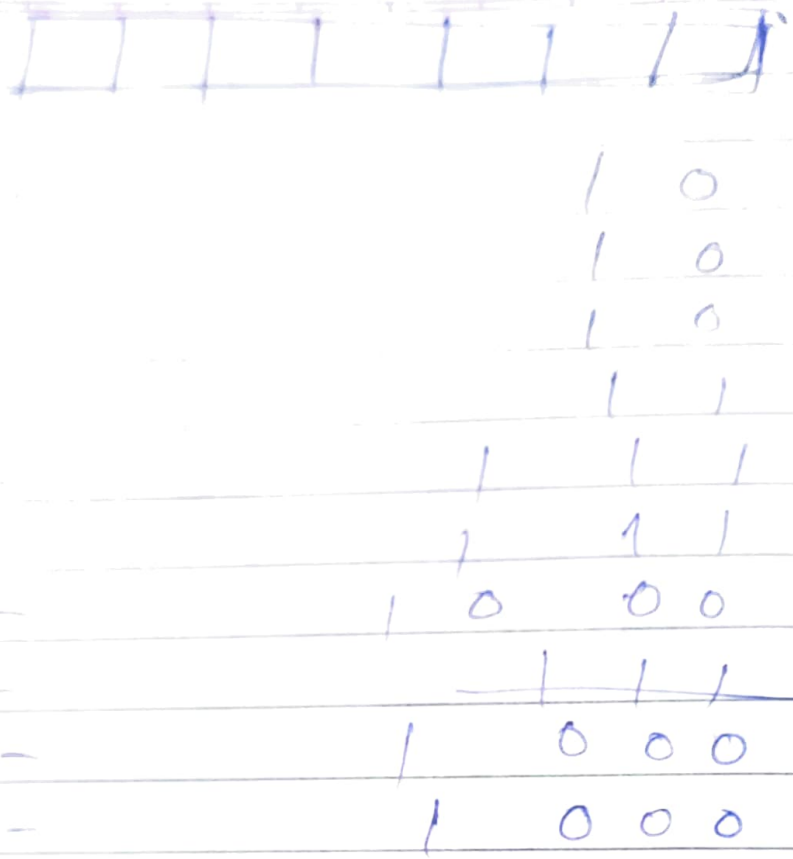
Idea: we know every no is appearing 3 times than the values of its bits will also be appearing 3 times

If we take an empty array :-

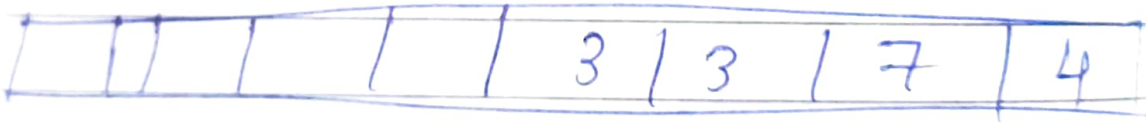
2 : - 10

Basically this array say that all the bits that is set means that are 1 past keep that adding those in this set array

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



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



imagine there was no. 3 so ↑ this will be something like [3][3][6] so obviously if it 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 becoming 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.

But here we're say is let's say add one of these nos that is some extra, so some of the digits may not be multiple of 3. we'll be having 7 & 4 because of 1 & 1

so if we just take the module % 3 " (the no which is repeating)

∴ % 3 = 0011 = 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 set bits will be % of 3.

" i.e remainder of 3 will be 0.
But if there is one extra no. i.e actually addition to the 3 multiple that we have so that won't be a multiple of 3."

Note: Not just for 3, we can do this for anyone e.g. for 3, 5, 7, ---- do it like - wise.

Amazone

ques

Q. find the nth magic no.

e.g. 1) 1 = 0 0 1
$5^3+5^2+5^1$ = 5 (magic no)

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

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

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

⋮

nth → [?]

lets say $n = 6$

(1) Convert the n into binary : 110
(2) we're gonna start 5' and keep on multiply these with these no.s 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 converte into binary internally

1) okay how we've 0 so leave by right shift 1 & move to next 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 or 1
n >> 1

0 * 5^1 + 1 * 5^2 + 1 * 5^3

Code

```
public static class magicNumber {
  public static void main(String[]) {
    {
      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(ans);
    }
}
```

Q. find no. of digits in base b

E.g $(6)_{10} = 1$

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

formula :

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

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

Step :-

$\log_b b \cdot \log_b a = x$

$= a = b^x$

Similarly

$\log_2 6 = x$

$6 = 2^x$. So this base thing represents the no. of digits

$$int(\log_b n) + 1$$

e.g. $\log_2 10 = 3.32$

$$10 = 2^{3.32}$$

{ 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?

→ $\log_2 10 + 1$

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)

pascals 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
```

Q find the sum of nth row?

⟹ Sum of each row = sum of all the

$nC_0 + \cdots$

i.e

Sum of each row = $nC_0 + nC_1 + nC_2 + nC_3 + \cdots = 2^n$

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

Ans ∴ $1 << (n-1)$ $1 \times 2^{n-1}$

VVIMP

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 "

E.g (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's should be there, rest everything should be 0

So that is know we figure out the given no is of power of 2 or not

as we know that
$100000 = 11111 + 1$ (n-1)

And if 0 100000
$\& \; 011111$
0

E.g (a)

```
  1 0 0 0 1 0
& 0 1 1 1 1 1
-------------
  0 0 0 0 1 0
```

: not a power of 2

Ex) 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 = ?

E.g $3^6 = 3 \times 3 \times 3 \times 3 \times 3 \times 3$ // O(n) — ① way

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

$3^6 = 3^{110_2}$ = ans = 1

ans = 1

n = 110
n & 1 → 0
if 0 then ignore
→ PTO →

« if 0 then ignore because we know 36 = 32+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 × 0 = 0 so it will not do anything Hence, we can ignore it.
& we can say that

= ans = ans × base

∴ ans = 9 & n = 1 now 1 >> 1 = 1

& obviously base - base × base

= 9 × 1 = 81

because it is doubling base every time.

because if we have

3^{110}

(base is doubling every time) : $3^4 \times 3^2 \times 3^0$

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

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

$\therefore$ 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 x base – do this always

even if we are sloping $3^0 \times 0$ that doesn't mean just 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 base = 6;

    int ans = 1;

    while (power > 0){
      if ((power & 1) == 1){
            ans = ans * base;
      }
      base = base * base;
      power = power >>1;
    }
    System.out.println (ans);
  }
}
```

$= O(\log(power))$

Q. given no. find the no. of set bit in it

E.g n = 9
n = 1001 Ans := 2

n & (-n) = 0001 → right most set bit

If we subtract with this n

n - [n & (n-1)] = 1000 — (1)

do keep doing this this till n is greater than 0.

"Because we are finding the right most bit and we're deleting it one by one"

E.g n = 1 0 0 1
& 1 0 0 0
1 0 0 0 — (1)
→ 8

8 & 7

```
  1000
& 0111
------
  0000
```

(2)

No. of setbits = no. of iterations (o log(n))

code:

```
public class Setbits{
psvm {
    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));
}
```