SCALER

Bit Manipulation - 1

TABLE OF CONTENTS

Amreshwar
Arunava Basak
Dhasthagiri Reddy
M S Haseeb Khan
Mahesh Baswaraj
Manohar A N
MOHAMMAD ALI
Pranjul Kesharwani
Shambhavi
Shivam Shiv
Shivanand Patil
shubhangi
Swapnali
Uddeepta Saikia

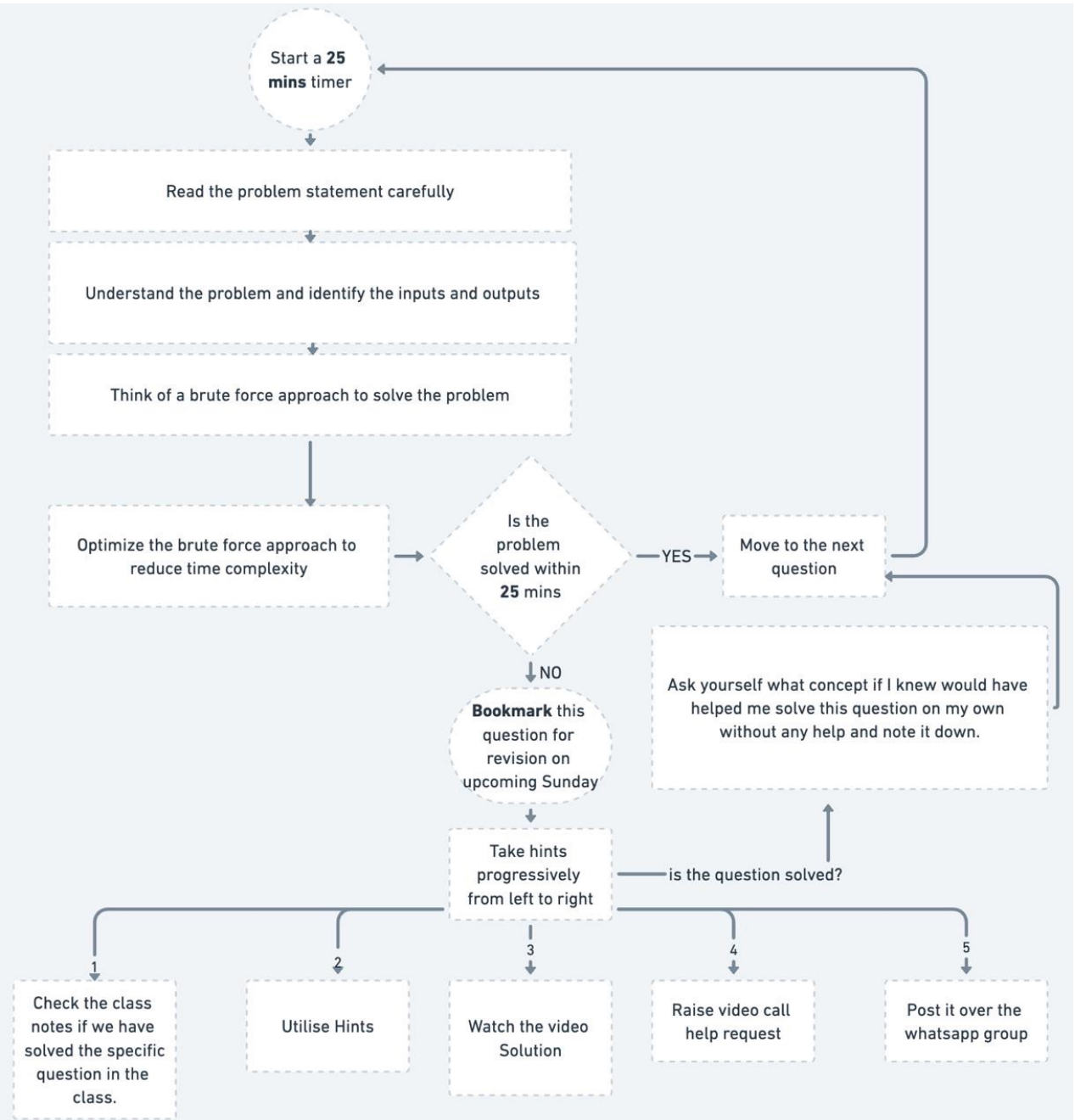
Rules

→ Q → QT

→ A → Private

→ Hands always on keyboard

Problem solving Framework



Bit-wise Operators : &, |, ^, ~, <<, >>

1 ⟶ T / ON / set / high voltage
0 ⟶ F / OFF / unset / low voltage

same same zero game

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

---

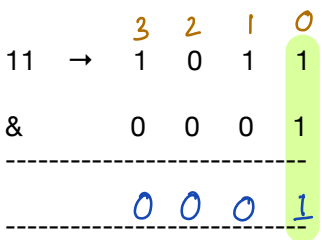
1 & 1 = 1
2 & 1 = 0
3 & 1 = 1
4 & 1 = 0
5 & 1 = 1
6 & 1 = 0

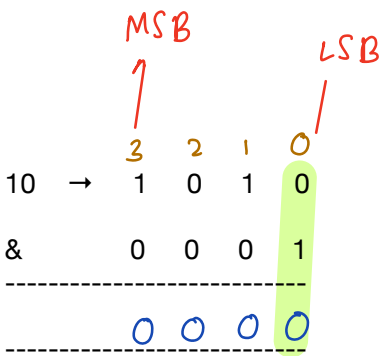
```
  11          100
& 01        & 001
----        -----
  01
```

```
  101
& 001
-----
  001
```

Basic Properties

1. **Even / Odd Number →**



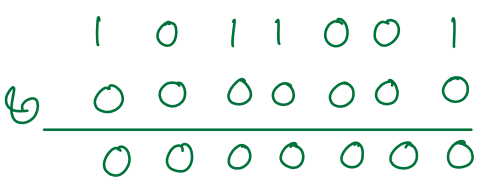


Observation

A & 1 == 1 ⟶ A is odd

A & 1 == 0 ⟶ A is even

2. **A & 0 →** 0



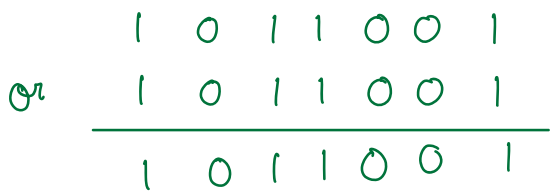
3. **A & A → A**

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

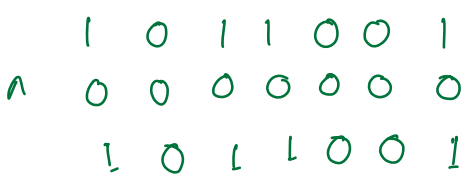
4. **A | 0 → A**

```
  1 0 1 1 0 0 1
| 0 0 0 0 0 0 0
-----------------
  1 0 1 1 0 0 1
```

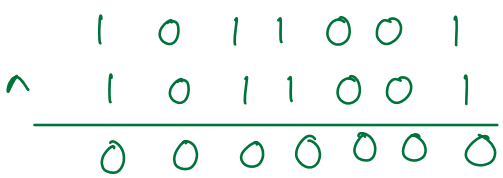
5. **A | A →** A



6. **A ^ 0 →** A



7. **A ^ A** → 0



Cumulative Property →

$$a \,\&\, b = b \,\&\, a$$
$$a \,\hat{}\, b = b \,\hat{}\, a$$
$$a \mid b = b \mid a$$

Associative Property →

$$a \,\&\, b \,\&\, c = (a \,\&\, b) \,\&\, c = a \,\&\, (b \,\&\, c)$$

$$a \,\hat{}\, b \,\hat{}\, c = (a \,\hat{}\, b) \,\hat{}\, c = a \,\hat{}\, (b \,\hat{}\, c)$$

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

< *Question- 1* > : Evaluate the expression: **a ^ b ^ a ^ d ^ b**

$$
\begin{aligned}
& a \wedge b \wedge a \wedge d \wedge b \\
\Rightarrow\ & a \wedge a \wedge b \wedge b \wedge d \\
\Rightarrow\ & (a \wedge a) \wedge (b \wedge b) \wedge d \\
=\ & (0 \wedge 0) \wedge d \\
=\ & 0 \wedge d \\
=\ & \underline{\underline{d}}
\end{aligned}
$$

< *Question- 2* > : Evaluate the expression: **1 ^ 3 ^ 5 ^ 3 ^ 2 ^ 1 ^ 5**

$$
\begin{aligned}
& 1 \wedge 3 \wedge 5 \wedge 3 \wedge 2 \wedge 1 \wedge 5 \\
\Rightarrow\ & (1 \wedge 1) \wedge (3 \wedge 3) \wedge (5 \wedge 5) \wedge 2 \\
\Rightarrow\ & 0 \wedge 0 \wedge 0 \wedge 2 \\
\Rightarrow\ & \underline{\underline{2}}
\end{aligned}
$$

< *Question* > : Given arr[N] where every element is present twice except one unique element. Find that unique element.

Input 1 2 3 1 2 3 5

Output 5

Idea ⟶ XOR everything duplicates will cancel each other and unique element remains

```
xor = 0
for i ⟶ 0 to N-1 {
    xor = xor ^ A[i]
}
print(xor)
```

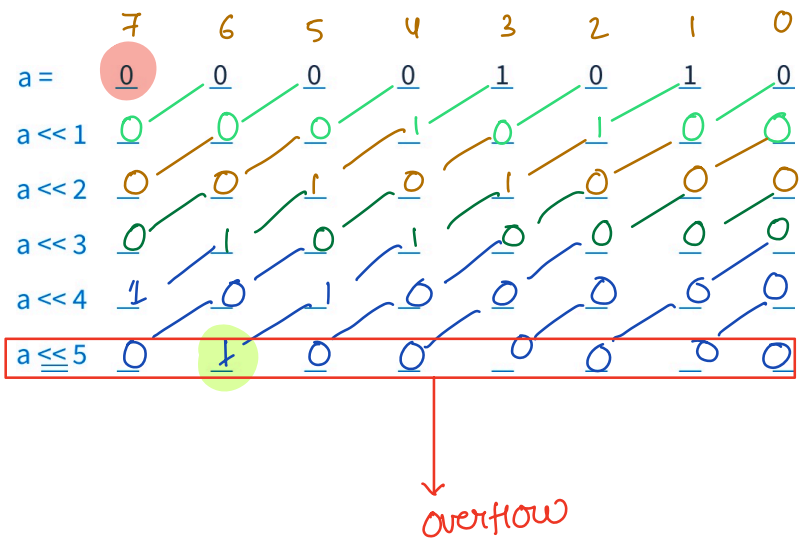
TC: O(N)

SC: O(1)

Left Shift Operator (<<)

Unsigned

Assumption
8 bit system



10 ×2 → 20 ×2 → 40 ×2 → 80 → 160 → 320 {Expected}

$10 * 2^0$
$10 * 2^1$
$10 * 2^2$
$10 * 2^3$

64 {actual}

max no. for 8 bit unsigned int = $2^8 - 1 = 255$

$a << 1 = a * 2$

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

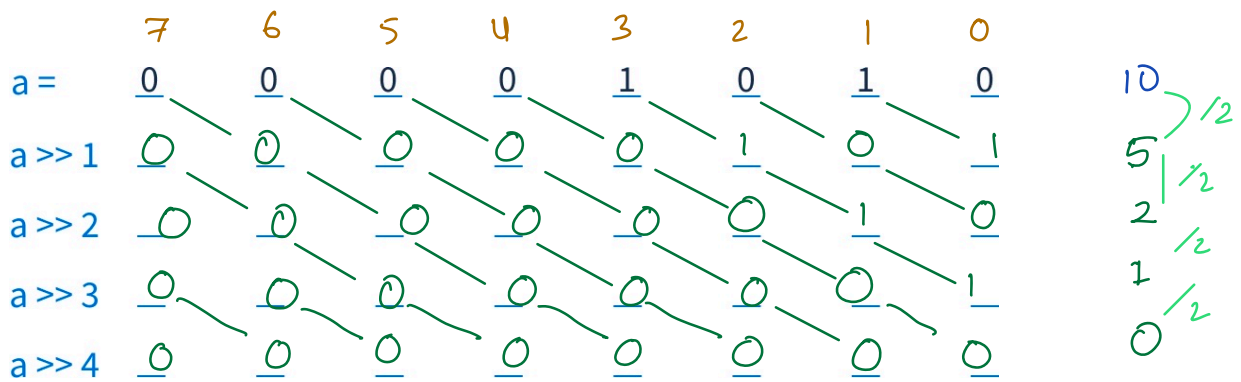
$a << 3 = a * 2^3$

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

$1 << n = 2^n$

Right Shift Operator (>>)

8 bit unsigned int



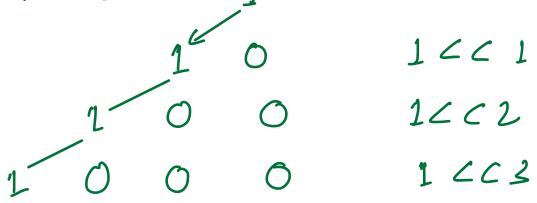
$a >> 1 = \frac{a}{2}$

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

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

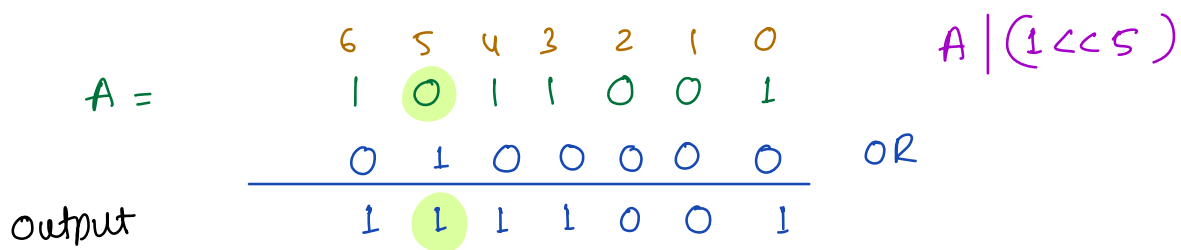
No overflow in case of right shift

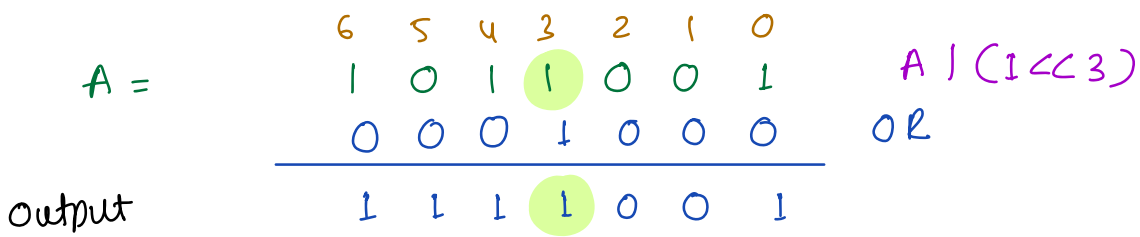
$1 << 3 =$ $1 << 1 =$ 1



Power of Left Shift Operator

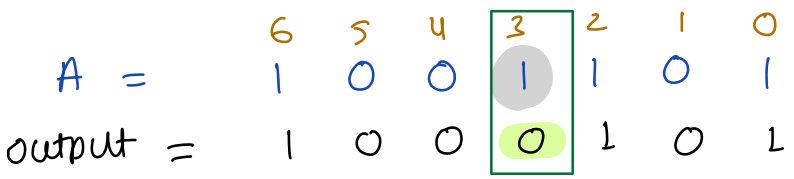
Q> Set 5^{th} bit of A

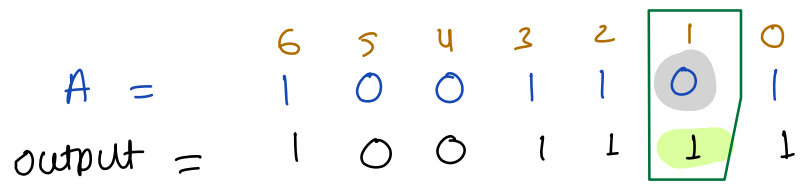


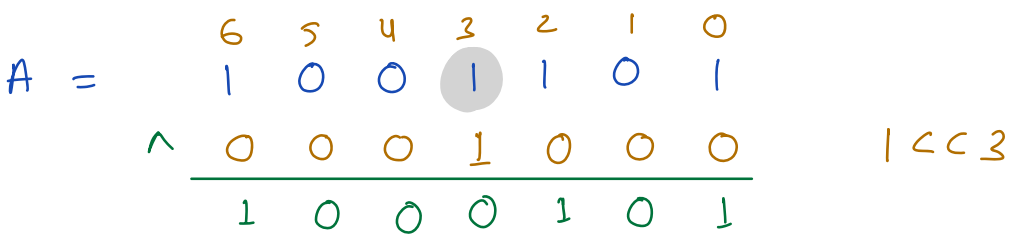


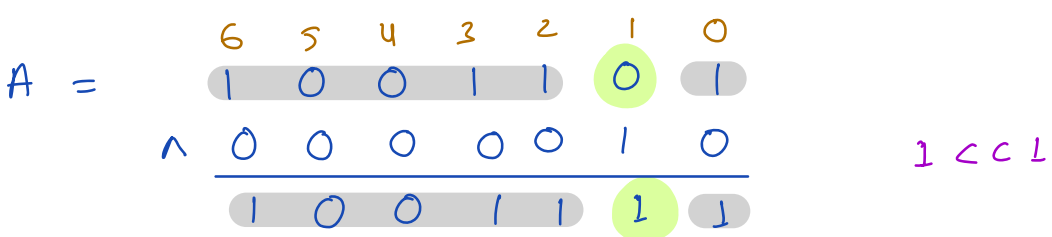
Set i^{th} bit in A = A | (1 << i)

Toggle the i^{th} bit



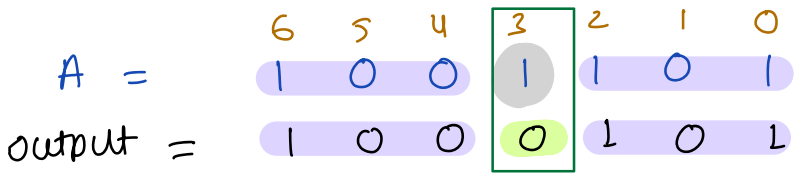


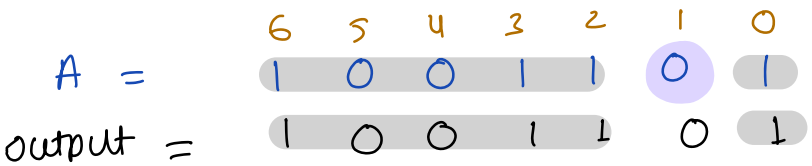


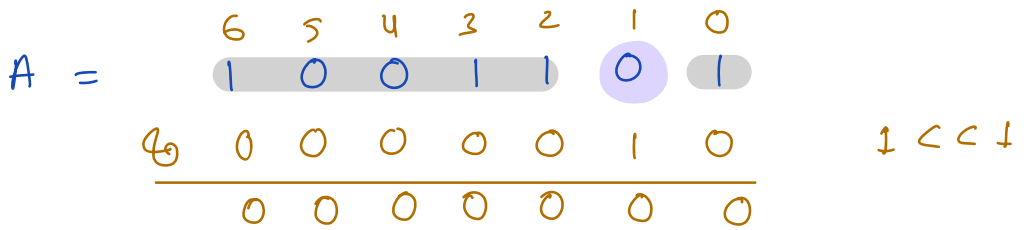


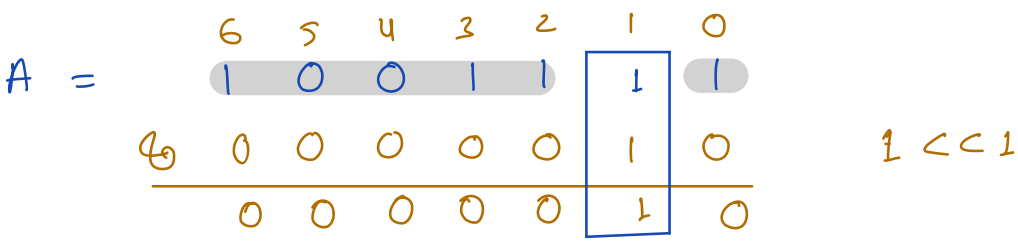
Toggle i^{th} bit = $A \wedge (1 << i)$

Unset i^{th} bit





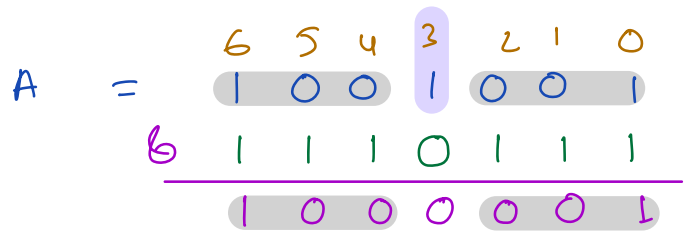


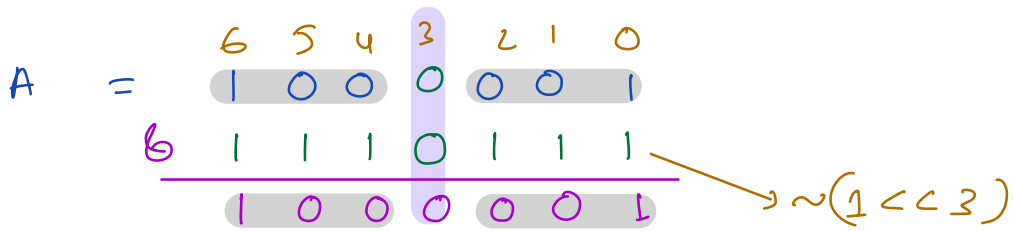


If $(A \,\&\, (1 << i)) > 0$ i^{th} bit is set in A.

```
X = A & (1 << i)
if (x > 0) {
    A = A ^ (1 << i)   // if ith bit is 1 or set
                       // we toggle to unset
}
```

Approach 2 >





Unset ith bit = A & (~(1 << i))

Approach 3 by Harshad

→ set ith bit

→ toggle ith bit

< *Question* > : Check whether ith bit is set or not.

```
          6 5 4 3 2 1 0
A =       1 0 0 1 1 0 1
```

Is 0th bit set ? ⟶ True

Is 1st bit set ? ⟶ False

```
          7 6 5 4 3 2 1 0
A =       1 0 1 1 1 0 0 1
```

& with 1 and if result == 1

```
          7 6 5 4 3 2 1 0
A =       1 0 1 1 1 0 0 1
      &   0 0 0 1 0 0 0 0
    ---------------------
X =       0 0 0 1 0 0 0 0
```

if (X > 0) return true
else return false

true if ith bit is set
false otherwise

Pseudocode

```
boolean checkBit ( A , i ) {
    X = A & (1 << i)
    if ( X > 0 ) return true
    else return false
}
```

TC: O(1)

< *Question* > : Given an integer N. Count the set-bits in N.

Example :

ans = 2

Idea ⟶ check for each bit in range 0 to 31
if the bit is set ⟶ cnt++

```
cnt = 0

for bit ⟶ 0 to 31 {
    if (checkBit(N, bit)) {
        cnt++
    }
}

print(cnt)
```

INT_MAX

TC : O(log(max))

~~TC : O(1)~~

SC : O(1)

∵ 31 is coming from the max no. of bit to store integer

range

int ⟶ ≃ -2^{31} to 2^{31}

long ⟶ ≃ -2^{63} to 2^{63}

log(max) = $\log(2^{31})$ = 31

log(max) = $\log(2^{63})$ = 63

N = 12

3	2	1	0
1	1	0	0

How to check if 0^{th} bit is set or unset?

if (N & 1 == 1) ⟶ 0^{th} bit is set
else 0^{th} bit is unset

	3	2	1	0	cnt
N	1	1	0	0	0
N >> 1	0	1	1	0	0
N >> 1	0	0	1	1	1
N >> 1	0	0	0	1	2
N >> 1	0	0	0	0	

```
cnt = 0

while ( N > 0 ) {
    if ((N & 1) == 1) { cnt += 1 }
    N = N >> 1    // N = N/2
}

print(cnt)
```

TC : O(log(N))
SC : O(1)

IRCTC

Scenerio

IRCTC (India's train ticketing system) wants to improve how it shows train options to its users. They've decided that trains which run more **frequently** should appear higher up in the search results. To figure this out, they look at a **28-day period** to see how often each train runs.

Problem

For **each** train, they've come up with a **special number**. This isn't just any number, though. If you were to write it down in binary form (which is like a special code of 0s and 1s), each of the 28 **digits** corresponds to a day in that **period**. A **'1'** means the train runs on that day, and a **'0'** means it doesn't.

Task

Your task is to help **IRCTC** by writing a program. Given a list **A** of these **special numbers** for different **trains**, your program should find the train that runs the most.

0 1 2

Input 1 : A = $[4369, 8738, 349525]$

Train No. (Index)	Binary Representation	# days
0	00000000000000010001000000001	3
1	00000000000000100010000000010	3
2	00000001010101010101010101101	11

Train with index 2 is most freq.

Idea 1 > max special no.

A = [8 , 7]

1 day — 1000 — 111 — 3days

Idea 2 > count no. of set bits

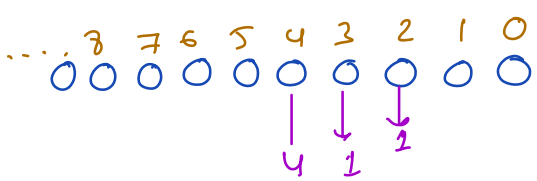
< *Question* > : Given three integer - A, B, C

Example : A = 4, B = 3, C = 2 $1 \le A, B, C \le 20$


```
ans = 0

for bit ⟶ C to C+B-1 {
    ans = ans | (1 << bit)
}

print(ans)
```



Edge : A = 20 B = 20 C = 20

Think what needs to be done here if return type is int