# Agenda :-

Bitwise Operators
↳ Properties
↳ few Problems .

$10 \rightarrow \underline{1010}$

no. → Binary

```
int a = 5;
int b = 10;
```

a (Bitwise operator) b

The operators which act upon the binary equivalent.

# Truth table of Bitwise Operators

xoR

| 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 |

## Basic And Properties

1) Even / odd number

10 → 1 0 1 0

9 → 1 0 0 1

_ _ _ _

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

In binary representation, if a number is even, then its least significant bit (LSB) is 0.

Conversely, if a number is odd, then its LSB is 1.

1) odd / Even Condn.

0th bit

```
A&1  →      x y z a b c 0/1
        &1  0 0 0 0 0   1
        -------------------
            0 0 0 0 0   0/1
        -------------------
```

A&1 → 1 (A is odd)
A&1 → 0 (A is even)

```
if ((A&1) == 1) {    // no' is odd
}
```

2) A & 0 = 0

A ⟶ $x\ y\ z\ \alpha\ \beta\ \gamma$<br>
& 0   0 0 0 0 0 0

0 0 0 0 0 0

3) A & A ⟶ A

A → 1 0 1 1 0<br>
& A → 1 0 1 1 0

A → 1 0 1 1 0

## OR properties

1) A | 0 = A

A → 1 0 1 1 1<br>
OR 0 → 0 0 0 0 0

1 0 1 1 1

2) A | A = A

A → 1 0 1 1 0<br>
A → 1 0 1 1 0

1 0 1 1 0

## XOR Properties

1) $A \wedge 0 = A$

$$\begin{array}{c} A \\ \downarrow \\ 1 \\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{c} A \\ \downarrow \\ 0 \\ 0 \\ \hline 0 \end{array}$$

$$\begin{array}{rccccc} A \rightarrow & 1 & 0 & 1 & 1 & 1 \\ \text{XOR } 0 \rightarrow & 0 & 0 & 0 & 0 & 0 \\ \hline & 1 & 0 & 1 & 1 & 1 \end{array}$$

2) $A \wedge A = 0$.

$$\begin{array}{c} A \\ \downarrow \\ 0 \\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{c} A \\ \downarrow \\ 1 \\ 1 \\ \hline 0 \end{array}$$

$$\begin{array}{rccccc} A \rightarrow & 1 & 0 & 1 & 1 & 0 \\ A \rightarrow & 1 & 0 & 1 & 1 & 0 \\ \hline & 0 & 0 & 0 & 0 & 0 \end{array}$$

## Commutative Property

↳ Order doesn't change the result.

$$a \,\&\, b = b \,\&\, a$$

$$a \mid b = b \mid a$$

$$a \wedge b = b \wedge a$$

## Associative Property

↳ grouping doesn't impact the overall result.

$$(A \,\&\, B) \,\&\, C = A \,\&\, (B \,\&\, C)$$

$$(A \mid B) \mid C = A \mid (B \mid C)$$

$$(A \wedge B) \wedge C = A \wedge (\underline{B \wedge C}).$$

## Ques

Evaluate the expression: $\underline{a \wedge b \wedge a \wedge d \wedge b}$

↓

$a \wedge b \wedge a \wedge d \wedge b$

↓

$\underline{a \wedge a} \wedge b \wedge b \wedge d$

$\underline{0 \wedge b} \wedge b \wedge d$

$\underline{b \wedge b} \wedge d$

↓

$\underline{0 \wedge d} \rightarrow \underline{d}$

## Ques

Evaluate the expression: 1 ^ 3 ^ 5 ^ 3 ^ 2 ^ 1 ^ 5

↓

$\underbrace{1 \wedge 1}_{0} \wedge \underbrace{3 \wedge 3}_{0} \wedge \underbrace{5 \wedge 5}_{0} \wedge 2$

↓

$\underline{2}$ .

# Left shift Operator (<<)

Let's say we have 8 bit numbers,

$a = \underline{10}$

| | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $a = 10$ = | | | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | | |
| | dis | | | | | | | | | | | |
| $a<<1$ = | | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | | => 20 | => $10\times2^1$ |
| | dis | | | | | | | | | | | |
| $a<<2$ = | | | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | => 40 | => $10\times2^2$ |
| $a<<3$ = | | | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | => 80 | => $10\times2^3$ |
| $a<<4$ = | | | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | => 160 | => $10\times2^4$ |
| | disc | | | | | | | | | | | |
| $a<<5$ = | | | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | => ~~320~~ 64 | |

$$\boxed{\begin{array}{l} a<<n = a*2^n \\ 1<<n = 2^n \end{array}}$$

(assuming no overflow)

$2^5$

$\underline{1<<5}$

# Right Shift Operator (>>)

| | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a = 20 | => | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | | |
| a >> 1 | => | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | discarded | => 10 |
| a >> 2 | => | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | dis | => 5 |
| a >> 3 | => | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | | => 2 |
| a >> 4 | => | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | => 1 |
| a >> 5 | => | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | => 0 |

$$\boxed{\begin{aligned} a >> n &= \frac{a}{2^n} \\ 1 >> n &= \frac{1}{2^n} \end{aligned}}$$

$$\underline{1 << 3} = \underline{2^3}.$$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

$0 \to 2^0$

$1 \to 2^1$

$0 \to 2^2$

$1 \to 2^3$

$1 \times 2^1 + 1 \times 2^3 \Rightarrow \underline{10}.$

* Set ith bit.

```
                 5 4 3 2 1 0
          N=     1 0 1 1 0 1
OR   1<<4 =      0 1 0 0 0 0
                 -----------
                 1 1 1 1 0 1
                 -----------
```

```
            5 4 3 2 1 0
       N=   1 0 1 1 0 1
OR  1<<3    0 0 1 0 0 0
            -----------
            1 0 1 1 0 1
            -----------
```

To set ith bit of a number

N = N | (1<<i)

# Toggle ith bit

```
                 5 4 3 2 1 0
          N=     1 0 1 1 0 1
xor  1<<4 =      0 1 0 0 0 0
     ------------------------
                 1 1 1 1 0 1
                 -----------
```

```
                 5 4 3 2 1 0
          N=     1 0 1 1 0 1
xor  1<<3        0 0 1 0 0 0
                 -----------
                 1 0 0 1 0 1
                 -----------
```

N= N ^ (1<<i)

# check bit at Particular idx.

$$
\begin{array}{llcccccc}
 & & 5 & 4 & 3 & 2 & 1 & 0 \\
 & N = & 1 & 0 & 1 & 1 & 0 & 1 \\
\text{And} & 1<<4 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
 & & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\longrightarrow 0
$$

$(N>>4)\&(1)$

$$
\begin{array}{llcccccc}
 & & 5 & 4 & 3 & 2 & 1 & 0 \\
 & N = & 1 & 0 & 1 & 1 & 0 & 1 \\
\text{And} & 1<<3 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
 & & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{array}
\longrightarrow \text{non-zero number}
$$

$N>>3 \rightarrow 0\,0\,0\,1\,0\,1$

$\& \; 1 \quad \underline{0\,0\,0\,0\,0\,1}$

```
if ((N & (1<<i)) == 0) {
        ith bit was unset
}
else {
        it was set.
}
```

```
function checkBit (N, i) {

    if ( (N& (1<<i)) == 0) {
        // bit was unset
            return false
    }
    else {
            return True;
    }
}
```

T.C → O(1)

S.C → O(1)

Ques Count no. of set bits in N.

N = 12 , → 1100 → Ans → 2 .

Approach 1:-

N=12

T.C → O(1)
S.C → O(1) .

```
function countBit (n) {
    ans = 0;
    for (i=0; i<32; i++) {
        if (checkBit (n,i)) { ans++ }
    }
    return ans;
}
```

Approach 2 :-

N = 1010 → 8 bit no

```
N =    0 0 0 0 1 0 1 0
& 1 =  0 0 0 0 0 0 0 1
       ---------------
       0 0 0 0 0 0 0 0
       ---------------

N>>1 =>  0 0 0 0 0 1 0 1
& 1      0 0 0 0 0 0 0 1
         ---------------
         0 0 0 0 0 0 0 1
         ---------------
```

```
N >> 2     0 0 0 0 0 0 1 0
& 1        0 0 0 0 0 0 0 1
           ---------------
           0 0 0 0 0 0 0 0
           ---------------

N >> 3     0 0 0 0 0 0 0 1
& 1        0 0 0 0 0 0 0 1
           ---------------
           0 0 0 0 0 0 0 1
           ---------------

N >> 4  =>  0 0 0 0 0 0 0 0  => 0
```

10
↓
5
↓
2
↓
↓0

```
ans = 0;
while (N > 0) {
    if ((N & 1) != 0)    } constant
    {
        ans++;
    }
    N = (N >> 1);        } const
}                  └──> /2 .
```

Theoretical →

T.C → O(log N) | T.C → O(1)

S.C → O(1) .

Ques

**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.

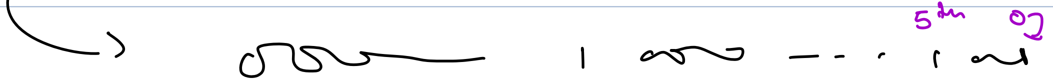
For **each** train, they've come up with a **special number**. This isn't just any number, though. If you were to write it down in binary form (which is like a special code of $0$s and $1$s), each of the $28$ **digits** corresponds to a day in that **period**. A **‘1’** means the train runs on that day,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

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 → 3   3 set   14 set bits

ex:-   A = [4369, 8738, 849525]

28 bit   28 bit   28 bit



2nd train.

Ques  Unset ith bit of a no' n.

N = 6,  → $\overset{2}{1}\ \overset{1}{1}\ \overset{0}{0}$    i = 2

→ 0 1 0 → 2.

$$\begin{array}{rccc} N = & 1 & 1 & 0 \\ 1<<2 \mid \wedge & 1 & 0 & 0 \\ \hline & 0 & 1 & 0 \end{array}$$

① check ith bit.

② if bit is set then do XOR.

T.C → O(1)
S.C → O(1).

```
func unset (N, i) {
    if (checkBit (N, i)) {
        N = N ^ (1<<i)
    }
    return N;
}
```

wrong
↓
read about it.

sign <

$\boxed{1}$ 1 0 1 → -5

$\boxed{0}$ 1 0 1 → 5.

# Ques Set Bits in a Range.

A group of computer scientists is working on a project that involves encoding binary numbers. They need to create a binary number with a specific pattern for their project. The pattern requires A 0's followed by B 1's followed by C 0's. To simplify the process, they need a function that takes A, B, and C as inputs and returns the decimal value of the resulting binary number. Can you help them by writing a function that can solve this problem efficiently?

A = 4, B = 3, C = 2 → 28 Ans.

=> 0 0 0 0 1 1 1 0 0

e.g

A = 4, B = 3, C = 2

7 6 5 4 3 2 1 0

N = 0 0 0 0 0 0 0 0 0 0 0 0 0

$C^{th}$ bit

A = 2, B = 4, C = 3

8 7 6 5 4 3 2 1 0

0 0 0 0 0 0 0 0 0

```
ans = 0;                    → B times.
for (i=0; i<B; i++) {
    N = setBit(N, C+i);
}
```

T → O(B) ≈ O(1).

A = 2, B = 4, C = 3 → 1111000

11 → 100

111 → 1000

1111 → 10000

11111 → 100000

10000 → 1111

1000000 → 111111

1 << B ⇒ 10000

((1 << B) - 1) ⇒ 1111

((1 << B) - 1) << C ⇒ 1111000