SCALER

Bit Manipulation - 1

TABLE OF CONTENTS

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

Truth Table

a	b	a&b	a|b	a^b	~a/!a
0	0	0	0	0	~0 = 1
0	1	0	1	1	~1 = 0
1	0	0	1	1	
1	1	1	1	0	

a & b : Returns TRUE when both A & B are set.

a | b : Returns TRUE when either A or B is set.

a ^ b : Returns TRUE when both A & B are different

Basic AND Properties

Even / Odd Number → Even no : LSB is 0
Odd no : LSB is 1

1. **A & 1** → 1 (odd)
   0 (Even)

```
11 → 1 0 1 1
&
1  → 0 0 0 1
-----------
     0 0 0 1  = 1
-----------
```

```
10 → 1 0 1 0
&
1  → 0 0 0 1
-----------
     0 0 0 0  = 0
-----------
```

2. **A & 0** → 0

```
9  → 1 0 0 1
&
0  → 0 0 0 0
-----------
     0 0 0 0  = 0
-----------
```

3. **A & A** → A

```
9  → 1 0 0 1
&
9  → 1 0 0 1
-----------
     1 0 0 1  = 9
-----------
```

Basic OR Properties

1. **A | 0 → A**

```
11 → 1 0 1 1
 | 
0  → 0 0 0 0
-------------------
     1 0 1 1 = 11
-------------------
```

```
10 → 1 0 1 0
 | 
0  → 0 0 0 0
-------------------
     1 0 1 0 = 10
-------------------
```

2. **A | A → A**

```
 9 → 1 0 0 1
 | 
 9 → 1 0 0 1
-------------------
     1 0 0 1 = A
-------------------
```

3. **A | 1 → A** (Odd No)
   A+1 (Even No)

```
11 → 1 0 1 1
 | 
1  → 0 0 0 1
-------------------
     1 0 1 1 = 11
-------------------
```

```
10 → 1 0 1 0
 | 
1  → 0 0 0 1
-------------------
     1 0 1 1 = 11
-------------------
```

Bit Manipulation-1

SCALER

Basic XOR Properties

1. **A ^ 0 → A**

```
11 → 1 0 1 1
0  → 0 0 0 0
---------------
     1 0 1 1  = 11
---------------
```

```
10 → 1 0 1 0
0  → 0 0 0 0
---------------
     1 0 1 0  = 10
---------------
```

2. **A ^ A → 0** V.V.V. Imp.

```
9 → 1 0 0 1
9 → 1 0 0 1
---------------
     0 0 0 0  = 0
---------------
```

Cumulative Property → The order of the operands doesn't affect the result of the operation.

$A \& B = B \& A$

$A \mid B = B \mid A$

$A \wedge B = B \wedge A$

Associative Property → Grouping of operands doesn't affect the result of the operation.

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

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

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

Quiz : 1

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

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

$= 0 \wedge 0 \wedge d$

$= 0 \wedge d = d$

Quiz : 2

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

$= (1 \wedge 1) \wedge (3 \wedge 3) \wedge (5 \wedge 5) \wedge 2$

$= 0 \wedge 0 \wedge 0 \wedge 2$

$= 0 \wedge 2 = 2$

ans : 2

Left Shift Operator (<<)

- Shifts bits of a no to left by specified no of positions.

A = 10	0	0	0	0	1	0	1	0	A = 10
A<<1	0	0	0	1	0	1	0	0	20 = A x 2^1
A<<2	0	0	1	0	1	0	0	0	40 = A x 2^2
A<<3	0	1	0	1	0	0	0	0	80 = A x 2^3
A<<4	1	0	1	0	0	0	0	0	160 = A x 2^4
A<<5	0	1	0	0	0	0	0	0	~~320~~ 64 = A x 2^5

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

$1 << n = 2^n$

MSB gets dropped & all other bits gets shifted to left by 1 position. 0 gets added in the end.

Range of 8 bits : [0 255]

NOTE: << of a no beyond the bit capacity of the datatype can cause overflow.

Right Shift Operator (>>)

- Shifts bits of a no to right by specified no of positions.

A = 10	0	0	0	0	1	0	1	0	$10 = 10/2^0$
A >> 1	0	0	0	0	0	1	0	1	$5 = 10/2^1$
A >> 2	0	0	0	0	0	0	1	0	$2 = 10/2^2$
A >> 3	0	0	0	0	0	0	0	1	$1 = 10/2^3$
A >> 4	0	0	0	0	0	0	0	0	$0 = 10/2^4$

LSB gets dropped & all other bits gets shifted to right by 1 position. 0 gets added in the front.

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

$$1 >> n = \frac{1}{2^n}$$

$$1 << 3 = 1 \times 2^3 = 8$$

Power of Left Shift Operator

1. AND Operator →

```
                          2  1  0
45      →   1  0  1  1  0  1
&
1 << 2  →   0  0  0  1  0  0
-----------------------------
            0  0  0  1  0  0
-----------------------------
```

```
                       4  3  2  1  0
45      →   1  0  1  1  0  1
&
1 << 4  →   0  1  0  0  0  0
-----------------------------
            0  0  0  0  0  0
-----------------------------
```

N & (1<<i) → 0 (if ith bit is UNSET in N)

N & (1<<i) → 1 (if ith bit is SET in N)

N & (1<<i) can be used to check if ith bit is SET or not.

2. OR Operator →

```
45      →   1   0   1   1   0   1
            |
1 << 2  →   0   0   0   1   0   0
-----------------------------------
            1   0   1   1   0   1
-----------------------------------
```

```
45      →   1   0   1   1   0   1
            |
1 << 4  →   0   1   0   0   0   0
-----------------------------------
            1   1   1   1   0   1
-----------------------------------
```

N | (1 << i) : Set ith bit

3. XOR Operator →

```
45      →   1   0   1   1   0   1
 ^
1 << 2  →   0   0   0   1   0   0
-----------------------------------
            1   0   1   0   0   1
-----------------------------------
```

```
45      →   1   0   1   1   0   1
 ^
1 << 4  →   0   1   0   0   0   0
-----------------------------------
            1   1   1   1   0   1
-----------------------------------
```

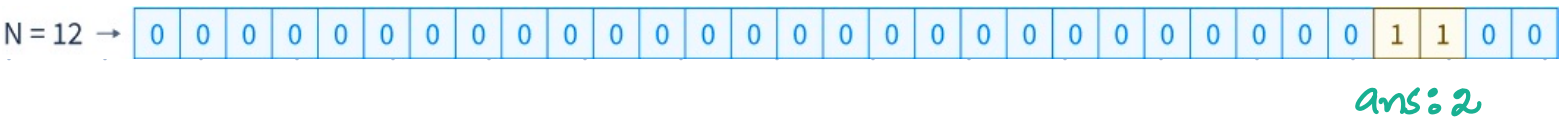
N ^ (1 << i) : Toggle the ith bit

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

```
bool checkbit(int N, int i){
    if(N & (1<<i) > 0)
        return true;
    else
        return false;
}
```

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

Example :



ans: 2

Iterate on all the bits and check if it is SET or not & update count.

```
int countbits(int N){
    ans = 0;
    for(i=0; i<32; i++){
        if(checkbit(i, N) == true)
            ans++;
    }
    return ans;
}
```

T.C : O(32) = O(1)

S.C : O(1)

Issue: It will work only for INT. Not for LONG etc.

0 0 0 0 _ _ _ _ _ _ 1

31 bits

Approach - 2: Using RIGHT SHIFT operator

N & 1 → 0 (IF LSB is 0)

N & 1 → 1 if (LSB is 1)

Approach : Check 0^{th} bit (N & 1) & right shift the no by 1 position.

Keep on doing this till N becomes 0.

			ans
N = 10	1 0 1 0	& 0 0 0 1 = 0	0
N >> 1	0 1 0 1	& 0 0 0 1 = 1	1
N >> 2	0 0 1 0	& 0 0 0 1 = 0	1
N >> 3	0 0 0 1	& 0 0 0 1 = 1	2
N >> 4	0 0 0 0 → STOP		

ans : 2

```
int CountSetBits(int N) {
    ans = 0;
    while(N > 0) {
        if((N & 1) == 1)
            ans++;
        N = N >> 1;    // N = N/2
    }
    return ans;
}
```

$N \to \frac{N}{2} \to \frac{N}{4} \to \frac{N}{8} \dots 1$

T.C : O(log N)
S.C : O(1)

< *Question* > : Unset the ith bit of N if it is **set.**

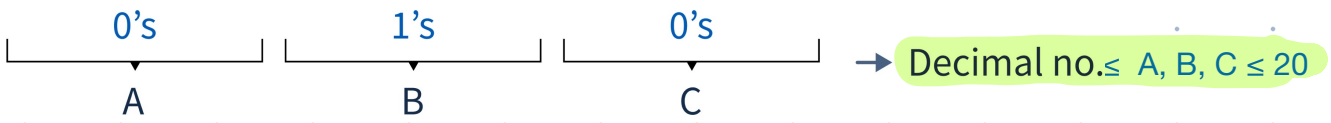
Check ith bit . If it is SET then toggle it.

```
if (checkbit (N, i) == true) {
    N = (N ^ (1 << i));
}
```

T.C : O(1)
S.C : O(1)

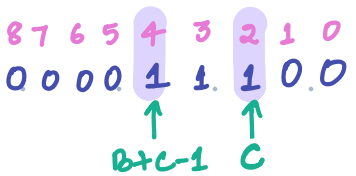
Set Bits in a Range

< Question > : Given three integer - A, B, C . It denotes A 0's followed by B 1's followed by C 0's. Return the resulting number.



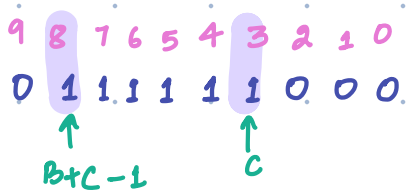
Example :

A = 4, B = 3, C = 2



ans : 28

A = 1, B = 6, C = 3



ans : 504

$[x \quad y] = B$

$y - x + 1 = B$

$y - C + 1 = B$

$y = B + C -$

Starting from C^{th} index → $(C+B-1)^{th}$ index set the bits.

```
long ans = 0;
for (i = c; i <= c+B-1; i++) {
    ans = ans | (1 << i);   // Set ith bit
}
```

A = 4 B = 3 C = 2 0000 11100

3+2-1 = 4

63 62 61 60 ... 7 6 5 4 3 2 1 0

ans: 0 0 0 0 ... 0 0 0 0 0 0 0 0

0 0 0 0 ... 0 0 0 1 1 1 0 0

1 1 1 1 ... 1 1 1 1 → 2×10^9

N = 4294967295.

log N = 32