Bit Manupulation

→ Decimal to Binary Conversion:-

$n = (7)_{10}$

2 | 7		
2 | 3		1
	1	1

→ 111 is 7

$n = (10)_{10}$

2 | 10		
2 | 5		0
2 | 2		1
	1	0

→ 1010 is 10

$n = (16)_{10}$

2 | 16		
2 | 8		0
2 | 4		0
2 | 2		0
	1	0

→ 10000 is 16

→ Binary to Decimal Conversion:-

$(101110)_2$ (5 4 3 2 1 0)

$0\times2^0 + 1\times2^1 + 1\times2^2 + 1\times2^3 + 0\times2^4 + 1\times2^5$

$\Rightarrow 2 + 4 + 8 + 32$

$\Rightarrow 46$

$(1000)_2$ (3 2 1 0)

1×2^3

→ 8

$(1111)_2$ (3 2 1 0)

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

$\Rightarrow 1 + 2 + 4 + 8$

$\Rightarrow 15$

Binary to Decimal.

0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7

1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

→ Decimal to Binary

n = 14

14 = 8 + 4 + 2

1000
0100
+ 0010
1110

Binary Addition → Right to left.

```
  1 1 1
  1 0 1 1.   → 11
+ 1 1 0 1.   → 13.
-------------
1 1 0 0 0
```

24

16 + 8 = 24

11000

Obs^n:-

$\Rightarrow 100000\ldots\ldots0 \Rightarrow 2^k$

All zeroes.

$\Rightarrow 111111111 \Rightarrow 2^k - 1$

Bitwise AND, Bitwise OR :-

&& , ||	& , |
↓ Logical and, ↓ Logical or.	↓ Bitwise And, ↓ Bitwise or.

Bitwise or.

0 | 0 => 0
0 | 1 => 1
1 | 0 => 1
1 | 1 => 1

Bitwise And.

0 & 0 => 0
0 & 1 => 0
1 & 0 => 0
1 & 1 => 1

⇒ Sout (5 | 9);

```
  0 1 0 1
Or 1 0 0 1
-----------
  1 1 0 1
```

↓ Convert to Decimal

13

Sout (5 & 11);

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

a b c d
or p q r s
1 0 1 1

b | q = 0

a | p = 1.
1 | 0 = 1
0 | 1 = 1
1 | 1 = 1.
x | 0 = x
x & 0 = 0

11 → 1 0 1 1

13 | 13 = 13.
13 & 13 = 13.
5 | 5 = 5
5 & 5 = 5
x | x = x
x & x = x

1 0 1 1	1 0 1 1	1 0 1 1
1 0 1 1	0 0 1 1	0 0 0 0
11 | 11 = 11	11 | 3 = 11	11 | 0 = 11.

1 0 0 1
1 0 1 0 — 9 | 2 = 11.

Data Types :-
byte :- -2^7 to $2^7 - 1$ ⟶ -128 to 127
Short :- -2^{15} to $2^{15} - 1$ ⟶ -32768 to 32767
int :- -2^{31} to $2^{31} - 1$
long :- -2^{63} to $-2^{63} - 1$

→ Binary Number me bits hote hai
1 bit ⟶ 2 numbers ko store kar sakta hai ⟶ 0 or 1.
~~1 byte = 8 bits~~

1 byte = 8 bits

0/1 0/1 0/1 0/1 0/1 0/1 0/1 0/1

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

2 × 2 × 2 × 2 × 2 × 2 × 2 × 2

$\Rightarrow 2^8 = 256$ options

int → Can store 2^{32} Numbers → -2^{31} to $2^{31}-1$

byte → 2^8 numbers → -128 to 127.

byte x = 17. 0 0 0 1 0 0 0 1

17 → 10001

byte x = 127. 0 1 1 1 1 1 1 1

Left Most bit is for Sign.

0 → +ve.

1 → -ve.

***

Bitwise XOR

∧ (XOR)

Even Number of Ones → 0

Odd Number of Ones → 1

1 ∧ 1 = 0

1 ∧ 0 = 1

0 ∧ 1 = 1

0 ∧ 0 = 0.

a|b|c|d == b|a|d|c.

Associative Hote Hai bitwise operator.

1's Complement, 2's Complement

flip All bits.

1 0 1 1 0 0 1

↓ 1's Complement

0 1 0 0 1 1 0

x, ~x = -x-1

6 ka is Compliment

⟹ -6-1 ⇒ -7,

2's Complement of x is -x.

2's Comp(x) = 1's Comp(x) + 1,

~ → tilde

` → backtick

nibble → Range → -8 to 7.
4 bits → 2^4 numbers

byte, short, int, long

How -ve Numbers Are Stored:-

nibble

0 0 0 0 → 0	1 0 0 0 → -0
0 0 0 1 → 1	1 0 0 1 → -1
0 0 1 0 → 2	1 0 1 0 → -2
0 0 1 1 → 3	1 0 1 1 → -3
0 1 0 0 → 4	1 1 0 0 → -4
0 1 0 1 → 5	1 1 0 1 → -5
0 1 1 0 → 6	1 1 1 0 → -6
0 1 1 1 → 7	1 1 1 1 → -7
+ve ✓	-ve ✗

Ese Store Nahi Hote

Ese Hote Hai.

1 0 0 0 → -8
1 0 0 1 → -7
1 0 1 0 → -6
1 0 1 1 → -5
1 1 0 0 → -4
1 1 0 1 → -3
1 1 1 0 → -2
1 1 1 1 → -1
+ve ✓

$-x = \sim x + 1$

nibble a = -6.

$-6 = \sim 6 + 1$

-ve Decimal No. Conversion to binary

6 → 0 1 1 0.

~6 → 1 0 0 1

+ 0 0 0 1

1 0 1 0

sign bit

$-5 = \sim 5 + 1$

5 = 0 1 0 1

~5 = 1 0 1 0.

+ 0 0 0 1

1 0 1 1

sign bit.

Behavior Of Operators:-

a → 0, 1

a ∧ 0 = a.

a ∧ 1 = flipped 'a'.

a | 0 = a.

a | 1 = 1.

To turn on a bit.

a & 1 = a

a & 0 = 0

To turn off a bit.

a * b → T.C = O(1)

↓

* => |, &, ^, >>, <<, ~

Right Shift

$a = 1\ 0\ 1\ 1\ 0\ 1\ 1 \quad (91)_{10}$

$a >> 2 = 0\ 0\ 1\ 0\ 1\ 1\ 0 \quad (22)_{10}$

↓ No of bits Shifted / Removed from Right.

$a >> 3 = 0\ 0\ 0\ 1\ 0\ 1\ 1 \quad (11)_{10}$

Left Shift :-

$a = 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1, \quad (91)_{10}$

$a << 2 = 1\ 0\ 1\ 1\ 0\ 1\ 1, 0\ 0 \quad (364)_{10}$

byte $x = 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \quad (102)_{10}$

$x << 3 = 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0 \quad (816)_{10}$

Q① :- Given 'n' find 2^n → T.C ⇒ O(1)

2^0	1
2^1	10
2^2	100
2^3	1000
2^4	10000
2^5	100000

Binary

$2^n = 1\underbrace{00\ldots0}_{n \text{ zeroes}}$

$2^n = 1 << n$ → T.C ⇒ O(1)

Q② :- Swap 2 numbers (XOR)

①
```
temp = a
a = b
b = temp
```

②
```
a = a ^ b
b = a ^ b
a = a ^ b
```

②
```
a = a + b
b = a - b
a = a - b
```

$x \wedge x = 0$

$x \wedge 0 = x$

XOR is Associative

$0 \wedge 0 = 0$, $1 \wedge 1 = 0$

$x = 9$

$x \wedge x \Rightarrow 1001 \wedge 1001 = 0000 \Rightarrow (0)_{10}$

$x \wedge 0 \quad 1001 \wedge 0000 = 1001$

$x \wedge 0 = x$

$x \wedge x = 0$

$a = a \wedge b$.

$b = a \wedge b$. $\Rightarrow (a \wedge b) \wedge b$
$\Rightarrow a \wedge (b \wedge b)$.
$\Rightarrow a \wedge 0 \Rightarrow a$.

$a = a \wedge b$ $\Rightarrow (a \wedge b) \wedge a$
$\Rightarrow b \wedge (a \wedge a)$.
$\Rightarrow b \wedge 0$.
$\Rightarrow b$. //

b	a
~~a∧b~~	~~a~~
~~a~~	~~b~~
a_b	b_b

M T W T F S S
Page No.: YOUVA
Date:

Check if k^{th} bit is set or Not → 1 hai ya nahi.

Bitmasking

```
                10  9  8  7  6  5  4  3  2  1  0
a    =           1  0  1  1  1  0  0  1  0  1  1      k = (3)
&
mask =           0  0  0  0  0  0  0  1  0  0  0   ⟶ 1<<3
                 0  0  0  0  0  0  0  1  0  0  0
```

```
                10  9  8  7  6  5  4  3  2  1  0
a    =           1  0  1  1  1  0  0  0  0  1  1      k = (3)
&
mask =           0  0  0  0  0  0  0  1  0  0  0   ⟶ 1<<3
                 0  0  0  0  0  0  0  0  0  0  0
```

if (a & mask == 0) ⟶ bit was off / Not set (false).
if (a & mask != 0) ⟶ set hai (true).
return (a & (1<<k) != 0);.

---

① Leetcode Q.No. (136) { Single Number } Need to return No. which occurs only once.

arr = {4, 1, 2, 3, 1, 4, 2} Ans

M-I :- Brute Force ⟶ T.C ⇒ $O(n^2)$, AS ⇒ $O(1)$

M-II :- Sorting ⟶ T.C ⇒ $O(n \log n)$, AS ⇒ $O(\log n)$

M-III :- HashMap ⟶ T.C ⇒ $O(n)$, A.S ⇒ $O(n)$

M-IV :- Using XOR Operator ⟶ T.C ⇒ $O(n)$, A.S ⇒ $O(1)$.

arr = {4, 1, 2, 3, 1, 4, 2} → Ans.

↳ 4 ^ 1 ^ 2 ^ 3 ^ 1 ^ 4 ^ 2

⇒ 4 ^ 4 ^ 1 ^ 1 ^ 2 ^ 2 ^ 3

⇒ 0 ^ 0 ^ 0 ^ 3

⇒ 0 ^ 3 ⇒ 3 //

M T W T F S S | Page No.: | Date: | YOUVA

Check if k^{th} bit is set or Not :- (Method II)

k=4 a = 1 0 1 1 1 0 0 0 0 1 1 (bit positions 10 9 8 7 6 5 4 3 2 1 0)

If this is (1) then No. is Odd.

If this is (0) then No. is Even.

b = a >> k;

b = 0 0 0 0 1 0 1 1 1 0 0

If (0) the No. is Even → off (false)

If (1) the No. is ODD → Set (True).

return (b%2 == 1);

Turn on / set the kth bit :- (|)

	10	9	8	7	6	5	4	3	2	1	0	
a ⇒	1	0	1	1	1	0	0	1	0	1	1	k = 5
(1<<k) ← mask ⇒	0	0	0	0	0	1	0	0	0	0	0	↓ on
	1	0	1	1	1	1	0	1	0	1	1	

a = a | mask.

→ a = a | (1 << k) //

Turn off / Clear the kth bit : (&)

	10	9	8	7	6	5	4	3	2	1	0	
a ⇒	1	0	1	1	1	0	0	1	0	1	1	k = 7
& mask ⇒	1	1	1	0	1	1	1	1	1	1	1	
	1	0	1	0	1	0	0	1	0	1	1	

a & 1 = a

a & 0 = 0.

mask = ~(1 << k)

a = a & mask.

Toggle the k^{th} bit. (XOR)

	10	9	8	7	6	5	4	3	2	1	0	
a =	1	0	1	1	1	0	0	0	0	1	1	k = 5
(m) mask = ^	0	0	0	0	0	1	0	0	0	0	0	toggle
	1	0	1	1	1	1	0	0	0	1	1	

0 ^ 1 → 1

1 ^ 1 → 0

m = 1 << k

→ a = a ^ m

→ turn off the
Rightmost Set bit

[a = 1000000000].

→ Method - 1 → find 'k'

a = 1 0 1 1 1 0 0 1 0 0 0

b = 1 0 1 1 1 0 0 (1) → odd No.

k = 0 1 2 3

```
b = a
k = 0.
while( b%2 == 0){
        b >> 1;
        k++;
}
```

T.C = O(log₂ n) / O(31).

→ Method - 2 :: [n & (n-1)] [n = n & n-1] ## T.C ⇒ O(1).

n = 32	n = 43	n = 42
↓	↓	↓
1 0 0 0 0 0	1 0 1 0 1 1	1 0 1 0 0 0
n-1 = 31	n-1 = 42	n-1 = 41
↓	↓	↓
0 1 1 1 1 1	1 0 1 0 1 0	1 0 0 1 1 1
n & n-1 = 0 0 0 0 0 0	n & n-1 = 1 0 1 0 0 0	n & n-1 = 1 0 0 0 0 0

n ⇒ 1 0 1 1 1 0 0 1 0 0 0

& n-1 ⇒ 1 0 1 1 1 0 0 0 1 1 1

1 0 1 1 1 0 0 0 0 0 0

② LeetCode Q. No 231 { Power of Two. }

Given 'n'. Check if it is a power of 2 or Not.

n = 64 → True

n = 60 → false

Method - 2 → '&' Operator

16 = 10000

& 15 = 01111

= 0 true

(n & n-1) == 0

64 = 1000000

& 63 = 0111111

= 0 true

TC ⇒ O(1)

Method - 1 → Recursion

n → n/2 → n/4 ... → 1

if(1) true

if(0) false

T.C = O(log₂ n)

n = 15

→ 1 1 1 1

& 1 1 1 0

1 1 1 0 != 0 false

③ LeetCode Q.No (191) { Number of 1 Bits }.

Page No.: YOUVA
Date:

Method :- (1) Brute Force :- #T.C => O(32)

count = 0

You will Create 32 Masks & you will check if the ith Bit is Set or Not

1 0 1 1 1 0 0 1 0 0 0

0 0 0 0 0 0 0 0 1 0 0

1 << i → i = 0 to 31

Method -(1) → Best

↳ Turning off the rightMost Set bit Again & Again

1 0 1 1 1 0 0 (1) 0 0 0.

Count:

∅ 1 0 1 1 (1) 0 0 0 0 0 0

1 0 1 (1) 0 0 0 0 0 0 0

1 0 (1) 0 0 0 0 0 0 0 0

(5) Ans.

(1) 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0

```
int count = 0;
while(n != 0){
    n = (n & (n-1));
    count++;
}
```

③ LeetCode Q.No (2220) { Minimum Bit Flips to Convert Number }

Ex(1). Given

start Goal.

13 18

a ∧ b = 0 if a & b both are 0.
if a & b both are 1

Need to return How much bit's we need to flip to Conver start to Goal.

→ Return (No. of Ones in start ∧ goal)

St => 14 Goal => 15

14 => 1 1 1 0

15 => ∧ 1 1 1 1

0 0 0 (1) Ans = (1)

13 => 0 1 1 0 1

18 = ∧ 1 0 0 1 0

1 1 1 1 1 Ans = 5 flip

Need to flip All the bit's in start

④ LeetCode Q.No. 342 { Power of Four } ↳

↳ 1, 4, 16, 64, 256 ----

M T W T F S S
Page No.: YOUVA
Date:

[M-1] → Brute Force (Recursion)

keep dividing by 4. → T.C = O(log n).

$n \longrightarrow \frac{n}{4} \longrightarrow \frac{n}{16} \cdots 1 \longrightarrow (\log_4 n)$

[M-2] → If Any number is power of four, then 100% it is a power of two as well but vice-versa is Not true.

→ All odd powers of 2 [2^n where n is odd] are not powers of 4.

2, 8, 32, 128 ---- these are not 4^k.

$4^k = (2^2)^k = (2^k)^2$ which is a perfect square

→ Check 'n' is perfect square or Not.

[M-3] $(1+x)^n = 1 + {}^nC_1 x^1 + {}^nC_2 x^2 + {}^nC_3 x^3 \ldots$

→ Any Power of four leaves a remainder '1' when divided by 3.

For Ex :- 1 % 3 = 1
4 % 3 = 1
16 % 3 = 1
64 % 3 = 1
256 % 3 = 1

So instead of Checking 'n' is perfect square or Not Check if n%3 == 1 or Not.

$4^n = (1+3)^n = 1 + {}^nC_1 \cdot 3^1 + {}^nC_2 \cdot 3^2 + {}^nC_3 \cdot 3^3 \ldots + {}^nC_n \cdot 3^n$

$= 1 + 3p$

---

⑤ LeetCode Q.No. 260 { Single Number III } ↳

In the given Array, 2 Numbers Appears Once, Rest All Appears twice.

→ Method - 1 :-

Using Hashmap.

* T.C → O(n)
* A.S → O(n)

→ Method - 2 :-

Sorting.

* T.C → O(n log n)
* A.S → O(n log n)

Method - 3 :- Bit Manipulation. [XOR & Rightmost Set Bit]

arr = { 1, 5, 7, 1, 8, 5, 9, 7, 2, 2 }

↳ 1^1 ^ 5^5 ^ 7^7 ^2^2 ^ 8 ^ 9.

XOR = 1

↳ 0 ^ 8 ^ 9.

8 ⇒ 1000

^ 9 ⇒ 1001

0001

⇒ 2 unique elements are different, so Atleast 1 bit will be different.

⇒ XOR of two Number is a number & in this number All once represent diff bits in a & b.

a & b.

```
  1011   a
^ 0101   b
---------
  1110   XOR
```

```
  0110   XOR
& 0101   XOR - 1.
---------
  0100
^ 0010
---------
```

I want this : 0010 (mask)

arr = { 1, 5, 8, 1, 8, 5, 9, 7, 2, 2 }

x = 7 ^ 9.

```
  0111
^ 1001
---------
x ⇒ 1110
```

```
x-1 ⇒ 1101
n ⇒ 1100
```

b1	b2
7	9
2	8
2	8
1	1
8	8
5	5
7	1
0	0

x = 1110

^ n = 1100

mask ⇒ 0010

ele & mask == 0 ? != 0

b1 != 0 — Remaining Are And

b2 == 0

⑥ LeetCode Q.No (78) { Sub sets }.

'n' elements in a set can form 2^n subsets.

[a,b] → {[ ], [a], [b], [a,b]}

[a,b,c] → {[ ], [a], [b], [c], [a,b], [a,c], [b,c], [a,b,c]}

* T.C ⇒ $O(n * 2^n)$
* S.C ⇒ $O(n * 2^n)$

n = 3 $2^n = 1 << n$

	a	b	c	a	b	c
0	0	0	0	_	_	_
1	0	0	1	_	_	c
2	0	1	0	_	b	_
3	0	1	1	_	b	c
4	1	0	0	a	_	_
5	1	0	1	a	_	c
6	1	1	0	a	b	_
7	1	1	1	a	b	c

bit-index ka Array-index se Map karna hai

⇒ Code :-

```
public List<List<Integer>> subsets(int[] nums){
    List<List<Integer>> ans = new ArrayList<>();
    int n = nums.length;
    int total = (1 << n);
    for(int num = 0 ; num < total ; num++){
        List<Integer> list = new ArrayList<>();
        for(int bitIdx = 0 ; bitIdx < n ; bitIdx++){
            int mask = (1 << bitIdx);
            if((num & mask) != 0)
                list.add(nums[bitIdx]);
        }
        ans.add(list);
    }
    return ans;
}
```

Q) LeetCode Q.No. (190) { Reverse Bits }

Page No. | YOUVA
Date:

n = 23

1 0 1 1 1

1 1 1 0 1

(23) → Ans

~~Only~~ 5 bits Only.

: Brute Force Solution (Not Good method)

Convert integer to Binary Array (Size 32)

2	23	
2	11	1
2	5	1
2	2	1
	1	0

11101

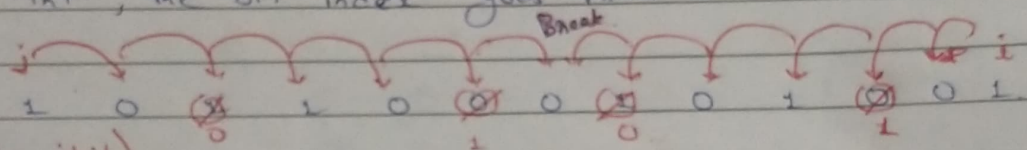
T.C => O(32)

A.S. => O(32)

* To much problem in this method :

Bit Manipulation Method

for 'int', the bit index goes from 0 to 31 i = 0, j = 31



if (jbit != ibit)

flip;

T.C => O(32).

A.S => O(1).

=> Code:

```
public int reverseBits(int n){
    int i=0, j=31;
    while(i<j){
        int iMask = (1<<i);
        int jMask = (1<<j);
        boolean ion = ((n & iMask) !=0);
        boolean jon = ((n & jMask) !=0);
        if(ion != jon){
            n ^= iMask;
            n ^= jMask;
        }
        i++; j--;
    }
    return n;
}
```

⑧ {XOR of Numbers in A given Range [a, b]}

[2, 9] ⇒ 2 ^ 3 ^ 4 ^ 5 ^ 6 ^ 7 ^ 8 ^ 9.

Page No.: YOUVA
Date:

Method - 1 :- Brute Force ⟶ loop [st to end]

↳ T.C ⇒ O(n)

XOR of 1 to n :- Observation

n =								
	1	=	1	9	=	1	n % 4 == 1	⇒ 1
	2	=	3	10	=	11	n % 4 == 2	⇒ n+1
	3	=	0	11	=	0	n % 4 == 3	⇒ 0
	4	=	4	12	=	12	n % 4 == 0	⇒ n
	5	=	1	13	=	1		
	6	=	7	14	=	15		
	7	=	0	15	=	0		
	8	=	8	16	=	16		

T.C ⇒ O(1)

```
xor (int n){
    if(n%4==1) return 1;
    if(n%4==2) return n+1;
    if(n%4==3) return 0;
    if(n%4==4) return n;
}
```

n = 77 ⇒ (1)

77 % 4 = 1

n = 62 ⇒ (63)

62 % 4 = 2

n = 87 ⇒ (0)

87 % 4 = 3

⇒ XOR from 3 to 9

⟶ 3 ^ 4 ^ 5 ^ 6 ^ 7 ^ 8 ^ 9.

⇒ [1 ^ 2] ^ [1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 ^ 7 ^ 8 ^ 9]

xor(a, b) ⇒ xor(1, a-1) ^ xor(1, b)

⑨ Leet Code Q. No. (1310) {XOR queries of a Subarray}

Ex:-

arr = {1, 3, 4, 8} ⟵ n-size

queries = {{0, 0}, {1, 2}, {0, 3}, {3, 3}} ⟵ m-size

{0, 0} ⟶ xor of 0 to 0 ; {1, 2} ⟶ xor of 1 to 2

Ans = {2, 7, 14, 8}

M-1 :- Brute Force

T.C = O(n * n) ⟶ (TLE error)

M-2 : using prefix Sum Concept:

Arr = {1, 3, 4, 8}.

pre = {1, 2, 6, 14}.

(1,3) xor batao

→ 1 ^ 1^3^4^8.

T.C = O(m+n)

(0,3) → 4

(0,0) → 1.