AGENDA:

- DSA: Bit Manipulations
  - Truth Table for Bitwise Operators
  - Basic Bitwise Operator Properties
  - Problem 1: Single Number
    - Left Shift Operator (<<)
  - Power of Left Shift Operator
    - Setting the i-th Bit
    - Toggling the i-th Bit
    - Checking the i-th Bit
    - Unsetting the i-th Bit
  - Problem 2: Count the total number of SET bits in N
  - Problem 3: Single Number 3

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

0 → F unset OFF low voltage

1 → T set ON high voltage

if both a & b are 0 else 1

NOT

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 if a & b are 1, 0 otherwise

same same zero game

---

power of 2s → 1, 2, 4, 8, 16 ....

2^0, 2^1, 2^2, 2^3, 2^4 .....

11 → $2^3 + 2^1 + 2^0$

8 + 2 + 1

1 0 1 1

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

2	22	0
2	11	1
2	5	1
2	2	0
2	1	1
	0	

1 0 1 1 0

$2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$16 + 4 + 2 = \underline{\underline{22}}$

bin(11) $\longrightarrow$

Basic Properties

1. **Even / Odd Number** $\rightarrow$ $A \,\&\, 1 == 1 \rightarrow$ odd

$A \,\&\, 1 == 0 \rightarrow$ even

11	$\rightarrow$	1	0	1	1
&		0	0	0	1
		0	0	0	1

10	$\rightarrow$	1	0	1	0
&		0	0	0	1
		0	0	0	0

2. **A & 0** $\rightarrow$ 0 $\forall$ A

3. **A & A** → A

$$\begin{array}{rr} A & 10110 \\ \& \ A & 10110 \\ \hline & 10110 \end{array}$$

4. **A | 0** → A

$$\begin{array}{rr} A & 10110 \\ | & 00000 \\ \hline & 10110 \end{array}$$

5. **A | A** → A

	A	1	0	1	1	0
|	A	1	0	1	1	0
		1	0	1	1	0

6. **A ^ 0** → A

	A	1	0	1	1	0
^		0	0	0	0	0
		1	0	1	1	0

7. A ^ A → 0

	A	1	0	1	1	0
^	A	1	0	1	1	0
		0	0	0	0	0

$2 \wedge 2 = 0$ $11 \wedge 11 = 0$

~~Cumulative~~ Property →

Commutative

A & B == B & A

A | B == B | A

A ^ B == B ^ A

Associative Property →

order in which operations are done doesn't matter

(A & B) & C == A & (B & C)

(A | B) | C == A | (B | C)

(A ^ B) ^ C == A ^ (B ^ C)

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

a ^ b ^ a ^ d ^ b

$\Rightarrow$ a ^ a ^ b ^ b ^ d

$\Rightarrow$ 0 ^ 0 ^ d

$\Rightarrow$ d

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

$\Rightarrow$ 1 ^ 3 ^ 5 ^ 3 ^ 2 ^ 1 ^ 5

$\Rightarrow$ 1 ^ 1 ^ 3 ^ 3 ^ 5 ^ 5 ^ 2

$\Rightarrow$ 0 ^ 0 ^ 0 ^ 2

$\Rightarrow$ 2

Value of 120 ^ 5 ^ 6 ^ 6 ^ 120 ^ 5 **is -** 0

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

Find that unique element.

A = [4 , 5 , 5 , 4 , 1 , 6 , 6]

output = 1

Bruteforce → Create a counter of A and key with frequency == 1 is my ans

```python
# Bruteforce
# Create a counter out of the given A itself and return the key with freq == 1

import collections
A = [7, 5, 5, 1, 7, 6, 1, 6, 4]

def single_number_1(A):
  counter = collections.Counter(A)

  for k, freq in counter.items():
    if freq == 1:
      return k

  return -1

print(single_number_1(A))

# TC: O(N)
# SC: O(N)
```

```python
# Optimised XOR ALL

A = [7, 5, 5, 1, 7, 6, 1, 6, 4]

def single_number_1(A):
  xor = 0

  for val in A:
    xor ^= val

  return xor

print(single_number_1(A))

# TC: O(N)
# SC: O(1)
```

```
4
```

---

ans = 5

A = 2 2 3 3 5 4 4

binary

MSB LSB

	2	1	0
2	0	1	0
2	0	1	0
3	0	1	1
3	0	1	1
5	1	0	1
4	1	0	0
4	1	0	0
	3	4	3

binary

MSB → ← LSB

	2	1	0
2	0	1	0
2	0	1	0
3	0	1	1
3	0	1	1
5	1	0	1
4	1	0	0
4	1	0	0
	2	4	2

binary

MSB → ← LSB

	2	1	0
2	0	1	0
2	0	1	0
3	0	1	1
3	0	1	1
5	1	0	1
4	1	0	0
4	1	0	0
	3	4	3
%2			
	1	0	1

every element appears

2 → $2x$ → 0
2 → $2x+1$ → 1

3 → $3x$ → 0
3 → $3x+1$ → 1

4 → xor all

A = 2 2 2 3 3 3 5 4 4 4

	2	1	0
2	0	1	0
2	0	1	0
2	0	1	0
3	0	1	1
3	0	1	1
3	0	1	1
4	1	0	0
4	1	0	0
4	1	0	0
	3	6	3

multiples of 3

	2	1	0
2	0	1	0
2	0	1	0
2	0	1	0
3	0	1	1
3	0	1	1
3	0	1	1
5	1	0	1
4	1	0	0
4	1	0	0
4	1	0	0
	4	6	4
%3	1	0	1

Left Shift Operator (<<)

8 bit rep

10

	7	6	5	4	3	2	1	0	val
a =	0	0	0	0	1	0	1	0	10
a << 1	0	0	0	1	0	1	0	0	20
a << 2	0	0	1	0	1	0	0	0	40
a << 3	0	1	0	1	0	0	0	0	80
a << 4	1	0	1	0	0	0	0	0	160
a << 5	0	1	0	0	0	0	0	0	64

overflow

Note ⟶ Not applicable for python

$\Rightarrow$ left shift by 1 $\Rightarrow$ multiply by 2 $= 2^1$

left shift by 2 $\Rightarrow$ multiply by 4 $= 2^2$

left shift by 3 $\Rightarrow$ multiply by $= 2^3$

⋮

left shift by n $\Rightarrow$ multiply by $= 2^n$

$1 << 3 = 1 * 2^3 = 8$

$\Rightarrow$

6	5	4	3	2	1	0
0	0	0	0	0	0	1

$\Rightarrow$ 0 0 1 0 0 0

8

Right Shift Operator (>>)

a =	0	0	0	0	1	0	1	0	10
a >> 1	0	0	0	0	0	1	0	1	5
a >> 2	0	0	0	0	0	0	1	0	2
a >> 3	0	0	0	0	0	0	0	1	1
a >> 4	0	0	0	0	0	0	0	0	0

int division

Right shift by 1 ⇒ divide by 2

Right shift by 2 ⇒ divide by 2^2

Right shift by 3 ⇒ divide by 2^3

Right shift by 4 ⇒ divide by 2^4

Right shift by 5 ⇒ divide by 2^5

⋮

Right shift by n ⇒ divide by 2^n

$10 >> 2 = \frac{10}{2^2} = \frac{10}{4} = 2$

$4 << 3 = 4 * 2^3 = 4 \times 8 = 32$

Power of Left Shift Operator

Break : 9 : 35

If you want to set i^{th} bit

1. OR Operator → $N \mid (1 << i)$

		4	3	2	1	0
$N =$	1 0	0	1	1	0	1
$\mid$ $1 << 4$	0 0	1	0	0	0	0
Final res	1 0	1	1	1	0	1

$N \mid (1 << 4)$

Setting a bit ⇒ making it 1

$$\boxed{\text{Set } i^{th} \text{ bit} \Rightarrow N \mid (1 << i)}$$

I want to toggle the i^{th} bit

toggle 3^{rd} bit

$0 \rightleftarrows 1$

	6	5	4	3	2	1	0
N	0	1	0	1	1	0	1
^ 1 << 3	0	0	0	1	0	0	0
	0	1	0	0	1	0	1

	6	5	4	3	2	1	0
N	0	1	0	0	1	0	1
^ 1 << 3	0	0	0	1	0	0	0
	0	1	0	1	1	0	1

$$\boxed{\text{Toggle } i^{th} \text{ bit} \Rightarrow N \wedge (1 << i)}$$

How to check if the i^{th} bit is 0 or 1 ?

	7	6	5	4	3	2	1	0
N =	1	1	0	1	0	1	1	0
& 1<<4	0	0	0	1	0	0	0	0
	0	0	0	1	0	0	0	0

.

	7	6	5	4	3	2	1	0
N =	1	1	0	0	0	1	1	0
& 1<<4	0	0	0	1	0	0	0	0
	0	0	0	0	0	0	0	0

Tell me if 4^{th} bit is 0 or 1 ?

if N & (1 << i) == 0 :
    i^{th} bit is 0
else
    i^{th} bit is 1

```
def check_bit ( N , i ) :
    return  N & (1<<i)  >  0
```

Unset ith bit

[47]

```
# Code to unset the i-th bit

N = 0b11001101

# Use whatever we have learnt above
# Unset 2nd bit in N

"""
if the ith bit is already 0/unset don't touch it
if the ith bit is set or 1 -> toggle it to make it 0
"""
def unset(N, i):
  if check_bit(N, i):
    return N ^ (1 << i) # toggle ith bit
  return N

N = 0b100110110
i = 12
pb(N)
pb(unset(N, i))
```

```
100110110
100110110
100110110
```

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

$N <= 10^9$

10^{18}

Example :

32 bit rep

N = 12 →	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
																												3	2	1	0

output → 2

for all the bit position call check-bit fn

```python
# Code to count the total number of SET bits in N
N = 0b1001101100110

ans = 0
for i in range(32):
  # if check_bit(N, i):
  #   ans += 1
  ans += check_bit(N, i)

print(ans)
```

range(32) → log (N)

check_bit(N, i) → O(1)

TC: O(log N)

int ≈ 2^{32} log

long ≈ 2^{64} log

- Single Element III

Given arr[N], every element repeats twice except for 2 elements. Find the two unique elements.

Eg: A[6] = { 3 6 4 4 3 8 } = 6, 8

A[4] = { 4 9 9 8 } = 4, 8

Bruteforce Hashmap

01010 01000 01001 01011 01100 00110 10001

A[12] = 10 10 8 8 9 9 11 12 12 6 6 17

xor all ⟶ 11 ^ 17 ⟶

4	3	2	1	0
0	1	0	1	1
^ 1	0	0	0	1
1	1	0	1	0

1st bit

11 & 17 were diff

	4	3	2	1	0
10	0	1	0	1	0
10	0	1	0	1	0
8	0	1	0	0	0
8	0	1	0	0	0
9	0	1	0	0	1
9	0	1	0	0	1
11	0	1	0	1	1
12	0	1	1	0	0
12	0	1	1	0	0
6	0	0	1	1	0
6	0	0	1	1	0
17	1	0	0	0	1

set

10
10
11
6
6

unset

8 8
9 9
12 12
17

	4	3	2	1	0
10	0	1	0	1	0
10	0	1	0	1	0
8	0	1	0	0	0
8	0	1	0	0	0
9	0	1	0	0	1
9	0	1	0	0	1
11	0	1	0	1	1
12	0	1	1	0	0
12	0	1	1	0	0
6	0	0	1	1	0
6	0	0	1	1	0
17	1	0	0	0	1

reset

unset

set

17

```python
# Code to find the two unique numbers

A = [10, 10, 8, 8, 9, 9, 11, 12, 12, 6, 6, 17, -1, -1, 0, 0]

# xor all
xor = 0
for val in A:
  xor ^= val
```

O(N)

```python
# xor will contain the xor of 11 ^ 17
diff = -1
for i in range(32):
  if check_bit(xor, i):
    diff = i
    break
```

logN

TC: O(N)

SC: O(1)

```python
pb(xor)
print(diff)

a = 0 # first unique
b = 0 # second unique

for val in A:
  if check_bit(val, diff):
    a ^= val # group set
  else:
    b ^= val # group unset
```

O(N)

```python
print(a, b)
```