Bit Manipulation

Content

- Basics OR XOR AND
- Left Shift & Right Shift
- Problems

```
class Interval {
    int start
    int end
}
```

A = [I_1 I_2 I_3 I_4 ... I_n]

Aist = A[i].start

Aist = A.get(i).start

create I ⟶ ans.add(new Interval(st, end))

---

last class → current. → Monday

65 ⟶ 66 ⟶ 75%

Bitwise Operations

1 ⟶ true / set / on
0 ⟶ false / unset / off

A	B	AND A&B	OR A|B	XOR A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

addition w/o carry

same same zero game

Q> 45 | 10

45 ⟶ 1 0 1 1 0 1
10 ⟶ 0 0 1 0 1 0

1 0 1 1 1 1

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

2	45	1
2	22	0
2	11	1
2	5	1
2	2	0
2	1	1
	0	

$1*2^5 = 32$
$+\ 0*2^4 = 0$
$+\ 1*2^3 = 8$
$+\ 1*2^2 = 4$
$+\ 1*2 = 2$
$*\ 1*2^0 = 1$

47

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

Properties

AND

4	—	100	5	101	
6	—	110	7	111	
8	—	1000	9	1001	
10	—	1010	11	1011	

`A & 1 = 0 / 1`

`A & 1 = 0` → number is even

`A & 1 = 1` → number is odd

```
 10  ⟶   1010
&  1  ⟶   0001
      ------
      =  0000  = 0
```

`A & 0 = 0`

`A & A = A`

```
 10  ⟶   1010
&10  ⟶   1010
      ------
      =  1010  = A
```

OR

`A | 0 = A`

`A | A = A`

```
  1010
| 0000
------
  1010  = A
```

XOR

`A ^ 0 = A`

```
  1010
^ 0000
------
  1010  = A
```

$A \wedge A = 0$

Commutative Property

$A \wedge B = B \wedge A$

$A \,\&\, B = B \,\&\, A$

$A \mid B = B \mid A$

Associative Property

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

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

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

Q1> $a \wedge b \wedge a \wedge d \wedge b$

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

$0 \quad 0$

$0 \wedge 0 \wedge d$

$0 \wedge d = d$

Q 2> $1 \wedge 3 \wedge 5 \wedge 3 \wedge 2 \wedge 1 \wedge 5$

$1 \wedge 1 \wedge 3 \wedge 3 \wedge 5 \wedge 5 \wedge 2 = 0 \wedge 2 = 2$

Q> Given int[] A. Every no. occurs twice except one number. Find that unique number.

1 3 5 3 2 1 5 = ans 2

Approach ⟶ XOR all elements.

Pseudo code

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

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

Left Shift (<<)

Left shift operator shifts the bits to the left by specified no. of positions.

A = 10

8 bit system.

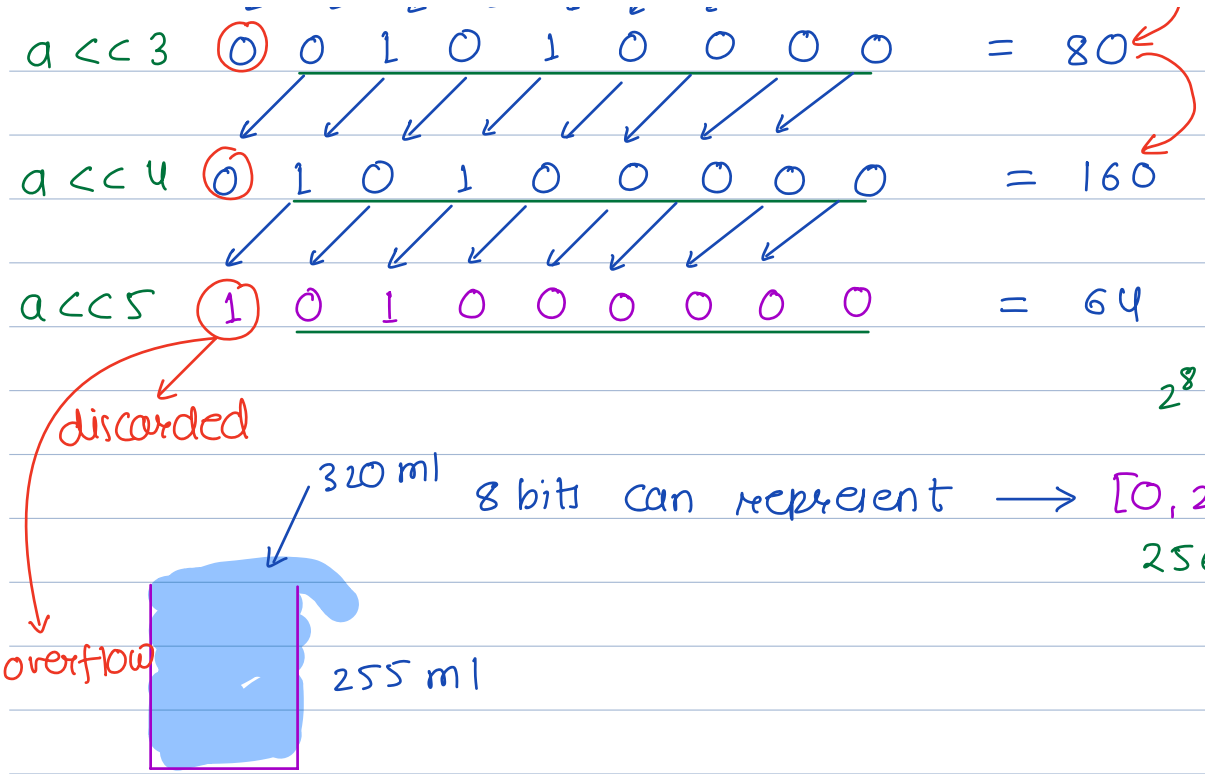
MSB ⟶ Most significant bit

LSB ⟶ Least significant bit

	7	6	5	4	3	2	1	0	
	0	0	0	0	1	0	1	0	= 10
discarded									
a << 1	0	0	0	0	1	0	1	0	0 = 20
a << 2	0	0	0	1	0	1	0	0	0 = 40

× 2

× 2



signed int = MSB is used to denote -ve value

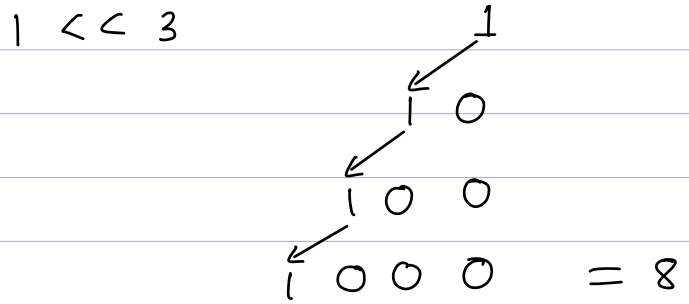
unsigned int = -ve values are not possible

$a << 0 = a * 2^0$

$a << 1 = a * 2^1$

$\vdots$

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



Right Shift >>

Right shift operator shifts the bits to the right by specified no. of positions.

a = 20 — 8 bit system

	7	6	5	4	3	2	1	0	
a =	0	0	0	1	0	1	0	0	= 20
a >> 1 =	0	0	0	0	1	0	1	0	= 10
a >> 2 =	0	0	0	0	0	1	0	1	= 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

/2 /2 /2 /2 /2

$a >> 0 = a/2^0$

$a >> 1 = a/2^1$

$a >> k = a/2^k$

Break: 22:40

a	1	1	1	1	1	1	1	1	⟶ 255
a >> 1 =	0	1	1	1	1	1	1	1	⟶ 127

Power of Left shift

`N | (1 << i)` = sets i^{th} bit in N

`1`

N = 45

	5	4	3	2	1	0
45	1	0	1	1	0	1
i = 3 1<<3 |	0	0	1	0	0	0
	1	0	1	1	0	1

⟶ 45

N = 45

	5	4	3	2	1	0
45	1	0	1	1	0	1
i = 4 1<<4 |	0	1	0	0	0	0
	1	1	1	1	0	1

⟶ 61

`N ^ (1 << i)` = flip/toggle i^{th} bit

N = 45

	5	4	3	2	1	0
45	1	0	1	1	0	1
i = 3 1<<3 ^	0	0	1	0	0	0
	1	0	0	1	0	1

N = 45

	5	4	3	2	1	0
45	1	0	1	1	0	1
i = 4 1<<4 ^	0	1	0	0	0	0
	1	1	1	1	0	1

N & (1 << i) = Checks if ith bit is set or unset

N = 45

	5	4	3	2	1	0	
45	1	0	1	1	0	1	
i = 3, 1<<3 &	0	0	1	0	0	0	
	0	0	1	0	0	0	= 1 << 3 = 8

N = 45

	5	4	3	2	1	0	
45	1	0	1	1	0	1	
i = 4, 1<<4 &	0	1	0	0	0	0	
	0	0	0	0	0	0	= 0

if N & (1 << i) == 0

ith bit in N is unset / 0

else

ith bit in N is set

---

check whether ith bit in N is set or unset

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

TC: O(1)

SC: O(1)

Q> Count the total no. of set bits in N. {int}

N = 12

1 1 0 0

Output = 2

Bits in int data type = 32

0 — 31

```
count = 0
for bit ⟶ 0 to 31 {
    if (checkBit(N, bit)) {
        count++
    }
}
print(count)
```

TC: O(1)

SC: O(1)

Assume you forgot the no. of bits in a datatype.

Idea 2 ⟶ keep checking 0th bit and right shift N by 1

N =	12	1 1 0 0	count = ~~0~~ ~~1~~ 2
	6	0 1 1 0	
	3	0 0 1 1	
	1	0 0 0 1	
	0	0 0 0 0	

Pseudocode

```
count = 0
while (N > 0) {
    if (N & 1 == 1) {  // check 0th bit
        count ++
    }
    N = N >> 1
}
print(count)
```

TC : O(log N)

SC: O(1)

$N \longrightarrow \frac{N}{2} \longrightarrow \frac{N}{4} \ldots\ldots 0$

log (N)

Approach 1 is same as approach 2

Q> Unset i^{th} bit in N

N = 6

1

1 1 0

If i^{th} bit is set, then unset it

else do nothing

Pseudocode

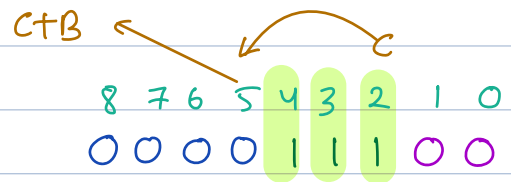
```
if ( N & (1 << i) != 0 )   // ith bit is set
|      N = N ^ (1 << i)    // toggle ith bit
}
print (N)
```

TC: O(1)
SC: O(1)

Q) 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 C as inputs and return decimal value of the resulting binary no.
Can you help them by writing a function that solves this problem efficiently?

$0 <= A\ B\ C <= 20$

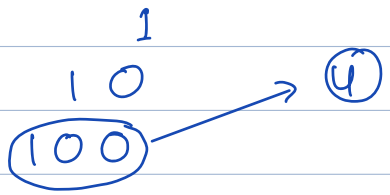
Input
A = 4
B = 3
C = 2




```
ans = 0                                   TC: O(B)
for i ⟶ C to C+B-1 {                      SC: O(1)
|      ans = ans | (1 << i)  // set ith bit.
}
```

Doubt session { Attendance is optional }

print (1 <<2)



Raise 2 HR per day