Welcome ☺

Agenda : Bit manipulation
Properties
2-3 questions.

Bitwise Operators.

`&` `|` `^` `~`

AND OR XOR NOT

same puppy same shame

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.

A & 1 = 1 ODD

A & 1 = 0 EVEN

eg: 0111 = 7
0001
& 0001 = 1

2. A & 0 = 0

3. A & A = A

BASIC OR Properties

$A \mid 0 = A$

$A \mid A = A$

$A \mid 1 = A \rightarrow$ if A is odd.

$A+1 \rightarrow$ if A is even

0111
0001

BASIC XOR Properties

$A \wedge 0 = A$

$A \wedge A = 0$

$A \wedge 1 = A-1 \rightarrow$ if A is odd.

$A+1 \rightarrow$ if A is even

0111
0001
^ 0110

0110
0001
^ 0111

Commutative Property

⇒ Order of operands does not affect result of bitwise operation.

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

$A \mid B == B \mid A$

$A \wedge B == B \wedge A$

Associative Property

→ Grouping of operands does not affect the result.

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

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

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

Quiz

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

$$\Rightarrow a \wedge a \wedge b \wedge b \wedge d \quad \text{// commutative prop.}$$

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

$$\Rightarrow 0 \wedge 0 \wedge d$$

$$\Rightarrow 0 \wedge d$$

$$\Rightarrow d$$

Quiz

~~$1 \wedge 3 \wedge 5 \wedge 3 \wedge 2 \wedge 1 \wedge 5$~~

$$\Rightarrow 2$$

Left Shift Operator (<<)

⇒ Shifts the bits of a number to the left by a specified number of positions.

⇒ It can be used to multiply a number by 2 raised to the power of specified no of pos^n

8 bit number

$a = 10 \Rightarrow 0\,0\,0\,0\,1\,0\,1\,0$

$7\,6\,5\,4\,3\,2\,1\,0$

a << 0	=	0 0 0 0 1 0 1 0	= 10
a << 1	=	0 0 0 1 0 1 0 0	= 20 (*2)
a << 2	=	0 0 1 0 1 0 0 0	= 40 (*2)
a << 3	=	0 1 0 1 0 0 0 0	= 80 (*2)
a << 4	=	1 0 1 0 0 0 0 0	= 160 (*2)
a << 5	=	discarded 0 1 0 0 0 0 0 0	= ~~320~~ 64

Overflow

⇒ Left shift a number beyond bit capacity of its datatype can lead to overflow.

⇒

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

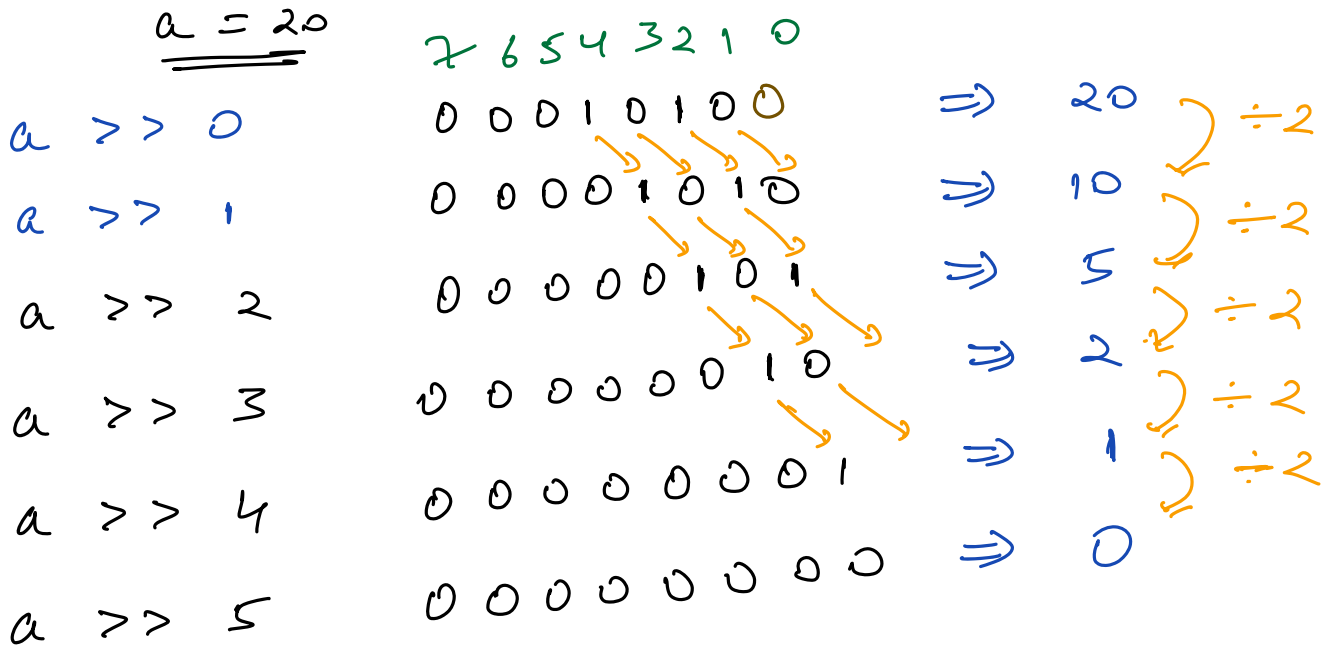
Right Shift Operator.

⇒ Shifts the bits of a number to the right by a specified number of positions.

⇒ Right shift operator divides the number by 2.

⇒ Overflow does not happen in right shifts.

a = 20

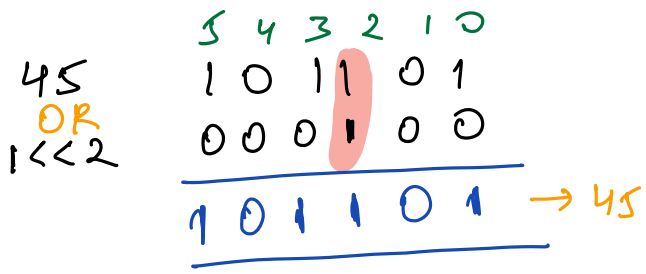


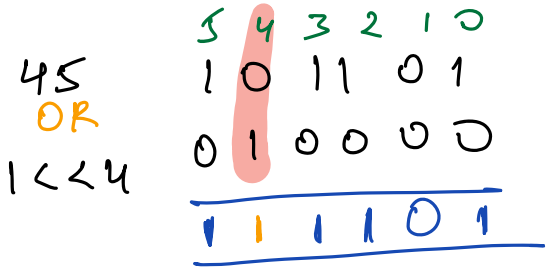
$a >> n$	=	$a / 2^n$
$1 >> n$	=	$1 / 2^n$

Power of Left Shift

1) SET i^{th} bit

N = 45 → 1 0 1 1 0 1

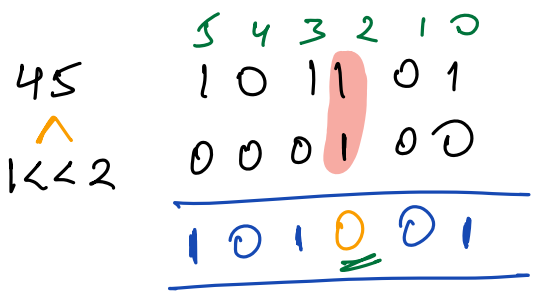


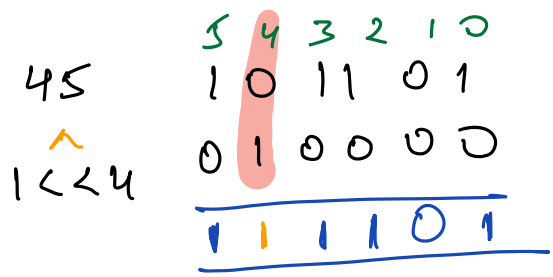


N | (1<<i) → if i^{th} bit is already set

→ N + (1<<i) → if i^{th} bit is unset

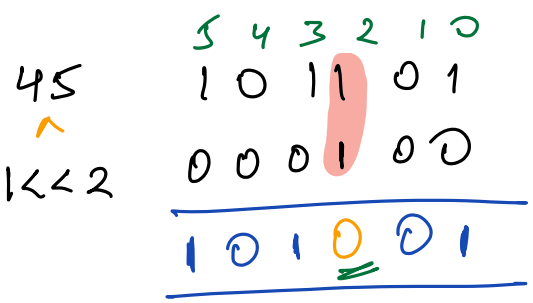
2) Toggle / FLIP i^{th} bit

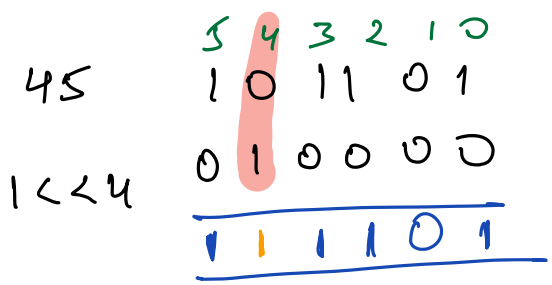




N ^ (1<<i) → Flip i^{th} bit

3) Unset a bit






```
if( checkBit( N, i ) ) → check if bit is set.
{
    N = N ^ (1<<i)  // Unset a set bit
}
```

Q1 Check if i^{th} bit is set or not

$0 \& 1 = 0$

$1 \& 1 = 1$

	5 4 3 2 1 0
45	1 0 1 1 0 1
AND 1<<2	0 0 0 1 0 0
	0 0 0 1 0 0 ≥ 0

	5 4 3 2 1 0
45	1 0 1 1 0 1
AND 1<<4	0 1 0 0 0 0
	0 0 0 0 0 0 $== 0$

1. Shift 1 to the i^{th} bit (1<<i)
2. $X == (N \& (1<<i))$

if ($X > 0$) → i^{th} bit is set

else → i^{th} bit is unset.

Q2 Given an integer N, count total no. of set bits in N.

eg N = 12

1100 ⇒ 2

App Iterate over all bits of integer (max. 32) and check whether it is set or not. If set, then increment ans by 1

```
func countBit(N)
{
    ans = 0
    for(i=0; i<32; i++)
    {
        if(checkBit(N, i)) ⇒ checks ith bit is set or not
        {                    ↓ True   ↓ False
            ans = ans + 1
        }
    }
    return ans
}
```

T.C ⇒ O(32) ≃ O(1)

App 2 We can use right shift operator.

Code

```
func countBit(N)
{
    ans = 0
    while(N > 0)
    {
        if(N & 1)
        {
            ans = ans + 1
        }
        N = (N >> 1)
    }
    return ans
}
```

T.C = O(log N)
≃ O(32)
for longer numbers

⇒ App 2 takes lesser time than App 1

Q Given A, B, C, create a pattern.
Pattern requires A 0's followed by B 1's followed by C 0's.
Write a func^n to return decimal value of this number.

$$0 \leq A, B, C \leq 20$$

eg: A = 4
B = 3
C = 2

8 7 6 5 4 3 2 1 0
0 0 0 0 1 1 1 0 0 = 28

long ans = 0
for (i -> C to B+C-1)
{
ans = ans | (1<<i)
}
return ans

App-2 → $2^B - 1$ 1000 → 0111

→ $(2^B - 1) << C$ → shift to left C times