24/09/2023

OPERATORS

* Arithemaric : +, -, *, /, %.
* Relational : >, <, >=, <=, ==, !=
* logical : &&, ||, !
* Bitwise : &, |, ^, <<, >>, ~
* others : sizeof, =, ----

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

int a=5, b=7

c = a & b

a = 0101
b = 0111
c = 0101 = 5.

c = a | b a=5, b=7

a = 0101
b = 0111
c = 0111 = 7.

a = 5, b=7 ^ (if same o/p = 0 els o/p = 1)

c = a ^ b

a = 0101 → 5
b = 0111 → 7
a^b = 0010 → 2

c = ~a → a = 5.

↓ int 4 bytes = 32 bits

a = 0 00 0101

(~a) = (1)111 1010

↓ ↓ -ve.

TO do in executor.

a & a = a	a & 0 = 0	a & 1 = even 0, odd 1 ?%
a | a = a	a | 0 = a	a | 1 → even (a+1) → odd (a)
a ^ a = 0	a ^ 0 = a	a ^ 1 → even (a+1) → odd (a-1)

a & 1 == a % 2.

a = 5 = 0101

a & 1 = 0001

0001 = 1 } odd (1)

a & 1 = 1 (Odd)

a & 1 = 0 (even)

a = 6 = 0110

a & 1 = 0001 = 0 → even.

0000

---

a = 5 = 0101

a | 1 = 0101 } odd

0001

0101 → 5

a = 6 = 0110

a | 1 = 0101

0

a | 1 = 0110 → 6

0001 → 1

0111 → 7 } even

a | 1 = a (odd)

a | 1 = a + 1 (even).

---

a = 5 = 101 → 5

a ∧ 1 = 001 → 1 } odd

100 (4)

a = 6 = 110 → 6

a ∧ 1 = 001

101 = 7

even

a ∧ 1 = a + 1 for (even)

a − 1 for (Odd)

Bitwise operators support the associative and the commutative properties.

* a | b = b | a
* a | (b | c) = (a | b) | c
* a & b = b & a.

left shift and Right Shift :-

left shift: <<

<<

7: 0 0 0 0 0 1 1 1 = 7

7<<1 : 0 0 0 0 1 1 1 0 = 7×2^1

(discarded)

7<<2 : 0 0 0 1 1 1 0 0 = 7×2^2

7<<3 : 0 0 1 1 1 0 0 0 = 7×2^3

discarded/ circled back will depend on the language or the compiler. In our session we treat it as discarded.

$N << i = N \times 2^i$ → (derived).

Right Shift :- >>

>>

20: → 0 0 0 1 0 1 0 0 → 20

20 >> 1 → 0 0 0 0 1 0 1 0 → $10 = \frac{20}{2^1}$

20 >> 2 → 0 0 0 0 0 1 0 1 → $5 = \frac{20}{2^2}$

By deriving we get :-

$N >> i = \frac{N}{2^i}$

problem :-

```
int powerof2(int N)
{

}
```

Brute fore method:- (Method 1)

```
int powerof2(int N)
{
    int res = 1
for(int i = 0; i<n; i++)
{
    res *= 2;
}
return res;
}
```

Using Bitwise operation:- (Method 2)

$$N * 2^i = N << i$$

$$a * 2^i = a << i$$

$$1 \times 2^N \Rightarrow a = 1$$

$$i = N$$

pow2(N) = 1 << N

```
int powerof2(int N)
{
    return (1 << N)
}
```

Eg: Imt powerof2(int 3)

```
{
    return (1 << 3)
}
```

2^3

$2^3 = 8$

0100

3 = 011

0001

1000

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

0001

1<<3 = 1000 = 8

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

range of integer : $[-2^{31} \quad 2^{31}-1]$

if we wanted to perform 2^i

then max value of i will be 30

so that 2^i will be in the range of int.

(1U << N)

↳ typecasting the int to unsigned

will increase the range of i from

30 to 31 as unsigned range is

$= [2^{32}-1]$

max of 31 can be allowed.

(1ll << N)

$2^{63}-1 = 62$

(1ULL << N)

$2^{64}-1 = 63$

Problem :-

check the i^{th} bit & return true if set

N	i	
5	2	0 1 0 1 → true
10	3	1 0 1 0 → true
8	2	1 0 0 0 → false
7	1	0 1 1 1 → true

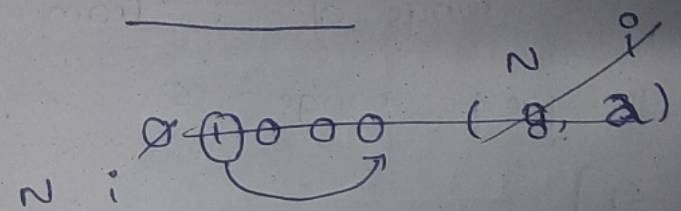
```
bool checkbit (int N, int i)
{
    return ((N >> i) & 1) != 0
        (or)
    return ((N >> i) % 2) != 0
}
```

Shift the ith bit of the Number to the end and check the Number is Odd or even

(N, i) = (5, 2)

0 1 0 1
  ↳ we should move 1 to end
So right by 2 times (i times)

0 0 0 1
& 0 0 0 1
---
① ≠ 0 hence set



(8, 2) → 1 0 0 0

0 0 1 0
& 0 0 0 1
---
0 0 0 0 = 0 (not set)

Alternate approach:-

(5, 2) = 0 1 0 1 instead of moving it bit to end
0 0 0 1

move the 1 from the end to i th position

N → 0 1 0 1 → N
1 << i → 0 1 0 0
(N & (1 << i)) → 0 1 0 0 (1 << i)
↳ ≠ 0 then set

```
return N & (1 << i) != 0
```

If we observe 1 << i == (N & (1 << i))

So we can also check as

```
return (N & (1 << i)) == (1 << i)
```

$1 \leq N \leq 10^9$, $0 \leq i \leq 30$

problem:-

checking how many bits are set in a given number.

i/p	o/p
N	
5 → 0101	→ 2 bits are set
16 → 10000	→ 1 bit is set
12 → 01100	→ 2 bits are set

Basic approach:-

```
int count = 0;
for(int i = 0 ; i <= 30 ; i++)
{
    if(checkbit(N, i))
    {
        count += 1;
    }
}
return count;
```

here the check happens for all 30 bits which is not needed. so to optimize it

Approach: 2

```
while(N != 0)
{
    if((N & 1) != 0)
    {
        count += 1
    }
    N >>= 1;
}
```

0 0 0 0 1 0 1 → 1st

(31st)

0 1 0 → 2nd ite

0 0 1 → 3rd.

0 0 0 → stop

=

approach : 3

N	(N-1)	N & (N-1)
5 : 0101	4 : 0100	0100 (4)
12 : 1100	11 : 1011	1000 (8)
16 : 10000	15 : 1111	00000 (0)
7 : 0111	6 : 0110	0110 (6)

```
int count = 0;
while (N != 0)
{
    N = N & (N-1);
}
return count;
```

problem :-

Generate a Number with only i^{th} bit & j^{th} bit :-

i	j
5	2
3	1
4	3
3	3

5th bit, 2nd bit.

→ ① 0 0 ① 0 0 → 36

→ 0 0 1 0 1 0 → 10

→ 0 1 1 0 0 0 → 24

→ 0 0 1_{2^3} 0_{2^2} 0_{2^1} 0_{2^0} → 8.

$N = 2^i + 2^j$ (if $i \neq j$)

but if $i = j$ then it gives wrong o/p

so | operation needs to be performed.

$2^5 \mid 2^2$ =

```
  100000
| 000100
--------
  100100  → 36
```

$2^3 \mid 2^3$ =

```
  001000
  001000
  001000   → 8
```

int numberwithij (int i, int j)

{

return $(1 << j) \mid (1 << i)$ → $0 \le i, j \le 30$

if $0 \le i, i \le 60$

}

if $(0 \le i, i \le 60)$

then $(111 << j) \mid (111 << i)$

problem :-

Generate a number which will have the x 1's followed by y 0's.

i/p	
x	y
3	2
2	2
4	3

```
  4 3 2 1 0
  1 1 1 0 0  → 28
    1 1 0 0  → 12
    3 2 1 0
0 1 1 1 1 0 0 0  → 120
  6 5 4 3 2 1 0
```

3+2-1 = 4

travel from i=y to i=x+y-1

x=3, y=2 ⇒ 11100 → starts from 2nd place and goes till 4th place (3+2-1)

```
for(int i=y; i<=x+y-1; i++)
{
    res += (1<<i);  (res += 2^i)
}
```

alternative approach :-

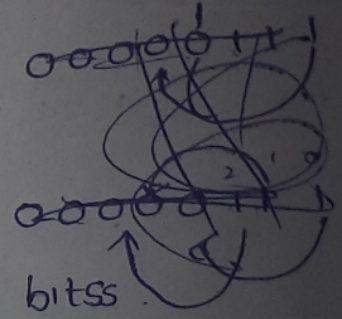
x	y
3	2

7 6 5 4 3 2 1 0

→ 0 0 0 1 1 1 0 0

$2^x = 8 : 2^3 \rightarrow 1000$

$2^x - 1 = 7 : (8-1) \rightarrow 0111$ ⟶



0 0 0 0 0 1 1 1 ⟶ shift by y bitss

0 0 0 1 1 1 0 0

$(2^x - 1) << y \rightarrow$ **((1<<x) - 1)<<y**

$a << i = a * 2^i$

$(2^x - 1)(2^y)$

$= 2^{x+y} - 2^y$

$= 2^x \cdot 2^y - 2^y$

(1<<x) (1<<y) - (1<<y)

= (1 << (x+y)) - (1<<y)