Bit manipulation.

01

- $(7)_{10} = (111)_2$ and
- $(13)_{10}$ = $(1101)_2$

```
2 | 13    1  ↑
2 | 6     0
2 | 3     1
  | 1
```

```
2 | 7   1  ↑
2 | 3   1
  | 1
```

and • $(1101)_2 = 1 + 0 + 4 + 8$
$= (13)$

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

$8 + 4 + 0 + 1$

$\Rightarrow (13)$ Ans

```
string convert2binary (int n) {
    res = " ";
    while (n != 1) {
        if (n % 2 == 1)
            res += '1'
        else
            res += '0'
        n = n/2;
    }
    reverse (res);
    return res;
}
```

TC = O($\log_2 n$)
SC = O($\log_2 n$).

```
int convert2decimal (string x) {
    int len = x.length(); p2 = 1,
    num = 0;
    for (i = len-1 → 0) {
        if (x[i] == '1')
            num = num + p2;
        p2 = p2 × 2;
    }
    return num;
}
```

TC = O(len)
SC = O(1).

- long long = 64 bits.

Now for 13 computer stores

0000 ⋯ 000   1101

28 bits   4 bits

- computer will not work well.

→ 1's complement

$(13) \rightarrow (1101)_2$

↓

$(0010)_2$

→ 2's complement

1) 1's complement
2) add 1 to it.

```
  0010
 +   1
 -----
  0011
```

→ AND, OR, XOR, SHIFT, NOT.

$x = 13 \& 7 = 5$

```
  1101
& 0111
------
  0101
```

all true = true
1 false = false

AND.

and $x = 13 | 7 = 15$

```
  1101
| 0111
------
 (1111)₂
```

all false = false
1 true = true

OR

and $x = 13 \wedge 7 = 10$

nos. of 1's → odd = 1
nos of 1's → even = 0

```
  1101
^ 0111
------
 (1010)
```

XOR

Now >> (right) = $\frac{x}{2^k}$ ⇒ $x >> k$

Now
Eg: $x = 13 >> 1 = 6$

1101 = 13 ⇒ $\frac{13}{2^1} = 6$

(000110) → 6

Eg: $x = 13 >> 2 = 3$

1101 ⇒ $\frac{13}{2^2} = 3$

$(11)_2 = 3$

Now for $x = -13$

so, 1101 → 13

1's complement = 0010
+ 1
2's complement 0011

Now << (left).

0 ... 1101
0 ... 11010 → 26

Now largest no.

0 1 1 1 ... 1 1

so, $(2^{31} - 1)$ = INT_MAX.

Now num << k

$num \times 2^k$

13×2^1

= 26

Now smallest no.

-2^{31}

1 00 ... 0
0 11111

```
  011..111
so,     +1
----------
..00 0000
```

-2^{31} = INT_MIN.

→ **swap two numbers.** ↪ by using XOR operators.

temp = a;
a = b;
b = temp;

eg. 5 ^ 5 = 0

```
  101
^ 101
-----
  000
```

eg. a = a ^ b
b = a ^ b
= (a ^ b) ^ b
= a.

now a = a ^ b.
= (a ^ b) ^ b̸ a
= b

→ **check if ith bit is set or not**

N = 13 i = 2 i = 1

$(1101)_2$ (1101)

yes NO.

now i.e by use of left shift or right shift operator.

eg.

N = 13 1101 1 << 2

i = 2 ↓ 10 → 100

```
  1101
& 0100
------
  0100
```

now
if ((N & (1 << i)) != 0)
yes
else
NO.

eg. N = 13
i = 1

1 << 1 → 10

```
 1101
 0010
 ----
 0000
```

NO not set.

→ **How to do this by right shift (>>)**

now N = 13
i = 2

3 2 1 0
1 1 0 1

1101 >> 2

```
so   1101
  &  0011
  -------
     0001
```

if ((N & (N >> i)) != 1)
NO
else
yes.

→ **set the ith bit**

N = 9
i = 2

3 2 1 0
1 0 0 1

do or operation

so, take 1 << (2) i → 10 → 100

```
 1001
 0100
 ----
 1101  → (13).
```

→ clearing the ith bit

N = 13 (1101) → (1001)

i = 2

So, do 1 << 2

and do negation.

So, 000100

~ 111011

Now 000111

& .. 111011

000011

→ remove last set bit

N = 12

1100

& 1011 → 11

1000 done.

1000

→ toggle the ith bit

N = 13

i = 2

1101

^ 0100 do XOR

1001

Now

1 << 2

1

10

100

N ^ (1 << i).

OBSERVATION:

N = 16	N = 40	N = 84
10000	101000	1010100
N = 15	N = 39	N = 83
01111	100111	1010011

→ check if no. be power of two or not

N = 16 N = 13 N = 32

yes NO yes.

So, if (N & N-1 == 0)

yes

else

NO.

• If there is 1 set bit

so answer is yes

else no.

So, Eg (16)

10000

01111

00000 yes done.

→ count no. of set bits

↳ By using STL

__builtin_popcount(n)

03 Minimum bit flips to convert number

Eg. start = 10 goal = 7

1010

0111

total we have to change 3 bits.

1010 start
^ 0111 goal
1101 → ans

- no. of set bits in ans will be your ans we.

Now start = 3 goal = 4

Eg. 011 100

total we have to change 3 bits.

011
^ 100
111 → ans

3 set bits so ans.

04 Power set

Eg. nums = [1 2 3]

[] [1], [2], [3]

[1,2] [1,3] [2,3]

[1,2,3]

total = 8.

no. of subsets = 2^n

1 << n

	2	1	0
[] ←	0	0	0
[3] ←	0	0	1
[2] ←	0	1	0
[2,3] ←	0	1	1
[1] ←	1	0	0
[1,3] ←	1	0	1
[1,2] ←	1	1	0
[1,2,3] ←	1	1	1

done.

0 → don't take
1 → take

indexing in array.

TC = N × 2^n

SC = 2^n × N

05 Single number I

nums = [4, 1, 2, 1, 2]

1 ^ 1 = 0

2 ^ 2 = 0

4 ^ 0 = 4

* start XOR ke do. *

06 single number II

Eg. nums = [2,2,3,2]

all nos are appears thrice except one. return that one.

→ use concept of buckets.

nums = [2,2,3,2]

ones – 0
twos – 0
threes – 0

- nums[i] will go to ones, if not in twos
- nums[i] will go to twos, if it is in ones.
- nums[i] will go to threes, if it is in twos.

Eg. 5 5 5 2 4 4 4

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

6, 3 → multiple of 3.

so, wo index pe set rhega jo 3 ka divisible nhi hoga. agar 3 ka divisible nhi hua toh waha set bit dalo.

07 single number III

nums = [2,4,2,14,3,7,7,3]

XOR = (2 ^ 2) ^ (14 ^ 4) ^ (3 ^ 3) ^ (7 ^ 7)
(2 ^ 2) → 0, (14 ^ 4) → 14 ^ 4, (3 ^ 3) → 0, (7 ^ 7) → 0

so, 14 = 1110
4 = 1100

- separate by buckets.

1110
0100

1010 → 10.

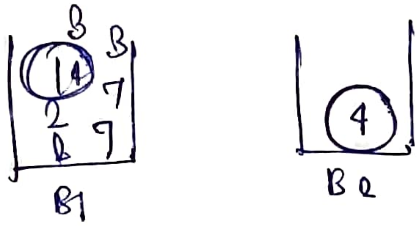
Now 10 = 1010
9 = & 1001

1000

Now 1000
^ 1010

0010 → 2 = (num & num−1) ^ num.

now [2 4 2 14 3 7 7 3] • 14 & 4 always be in diff. bucket



turned 1 turned 0.

08 XOR of numbers in given range. | now solve (1 to e)

N = 4 1 ^ 2 ^ 3 ^ 4 = 4

```
func (N) {
    if (N % 4 == 1) return 1
    else if (N % 4 == 2) return N + 1
    else if (N % 4 == 3) return 0;
    else return N;
}
```

09 Divide two integers.

eg 3 × 7

$3 \times (2^2 + 2^1 + 2^0)$

$(3 \times 2^2) + 3(2^1) + 3(2^0)$

12 6 3

so, 21 = 3 × 4 + 3 × 2 + 3 × 1

(3 × 4) + 6 + 3

(12) + 6 + 3

= (21)

so,
22
− 12
10
− 6
4
− 3
(1)

$3 \times 2^0 = 3$

$3 \times 2^1 = 6$

$3 \times 2^2 = 12$

$3 \times 2^3 = 24$