1. Swap all Even and odd index bits :→

Swap bits of Even and odd index

14 bits

num :→ 1 0 1 1 0 0 1 0 1 1 0 1 1 0

14 13 12 11 10 9 8 7 6 5 4 3 2 1

Even bits

result after swapping → 0 1 1 1 0 0 0 1 1 1 1 0 0 1

14 13 12 11 10 9 8 7 6 5 4 3 2 1

1-based index

1. Preserve odd index bits in n1
2. Preserve Even index bits in n2
3. Make A right shift in n2

   Even index → odd Even in backward dir

4. Make a left shift in n1

   odd index → even index in forward dir

5. Result = n1 | n2

   ↳ OR operator

Eg →

7 bits

0 1 1 0 1 0 1 0

8 7 6 5 4 3 2 1

after swapping →

1 0 0 1 0 1 0 1

8 7 6 5 4 3 2 1

n = 1 0 1 1 0 0 1 0 1 1 0 1 1 0

(indices) 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0 1 1 1 0 0 0 1 1 1 1 0 0 1

$\underline{a \,\&\, 1 = a}$

↳ we can preserve 'a' using and

① Preserve odd index

n = 1 0 1 1 0 0 1 0 1 1 0 1 1 0

✓ mask1 = 0 1 0 1 0 1 0 1 0 1 0 1 0 1

n1 = n & mask1 = 0 0 0 1 0 0 0 0 0 1 0 1 0 0

discard

1 at odd index. Rest all are '0'. odd index bits are preserve

② Preserve Even index

n = 1 0 1 1 0 0 1 0 1 1 0 1 1 0

mask2 = 1 0 1 0 1 0 1 0 1 0 1 0 1 0

n2 = n & mask2 = 1 0 1 0 0 0 1 0 1 0 0 0 1 ~~0~~ → Even index bits are preserve.

discard

Even index   odd index

After swapping

~~③~~ left shift in n1

↳ odd index bits are now become present at even index

④ Right shift in n2

↳ Even index bits are now become present at odd index

⑤ Result n1 | n2.

left shift

1 one

n1 = 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0

right shift

n2 = 0 1 0 1 0 0 0 1 0 1 0 0 0 1

① 2 swap

Result n1 | n2 = 0 1 1 1 0 0 0 1 1 1 1 0 0 1 mask1 = 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01

int mask1 = 0X 5 5 5 5 5 5 5 5;

5 in hexadecimal = 0101

mask2 = 1010 1010 1010 1010 1010 1010 1010 1010

1010 is equal to 'A' in hexadecimal

int mask2 = 0X AAAA AAAA

NOTE: Concern of odd and Even length is managed because of size of mask, we make mask with use of all 32 bits.

Sum of Bit Difference of All Pairs:

Pair $n_1 \leftrightarrow n_2$ — Sum of bit diff.

$$\left[\begin{array}{l} n_1 = 1\ 1\ 0\ 1\ 0\ 1 \\ n_2 = 1\ 1\ 0\ 0\ 1\ 0 \end{array}\right.$$

↳ 3 increment in difference count

Element (integers) ⇒ a, b, c, d.

Total pairs

c_1	c_2	c_3	c_3	----
a b	b a	c a	d a	
a c	b c	c b	d b	
a d	b d	c d	d c	

$$\text{Result} = \sum_{i=0}^{---} c_i$$

Sum of bit diff. ?

Bruteforce → $O(n^2) = O(32\,n^2)$ ✗

allowed $O(n) = O(32n)$

Pattern grasping.

D.S. (math) < 10+2 (math)

Logic — logic + f

virtual contest

[(4) → (2) solve] learning, confidence ↑

(2) unsolved → learning

solution

L.S too ↑

(2) — concept

Discussion

Solution (20)

(30) → (20)

↳ 3 → solve

1 → unsolve

(10) → learning

a →	1	0	1	1	1
b →	1	1	0	0	1
c →	1	0	1	0	1
d →	1	1	0	1	0

Complexity = $\underbrace{32 \times n}$ ≡ $O(n)$

0th index

ON bits: a, b, c

OFF bit: d

3 1

pairs

ad → single bit diff.
bd → " " "
cd → " " "

vice versa

→ da
→ db
→ dc

total increment = $\underbrace{3 \times 1}_{\text{pair}} \times \underbrace{2}_{\text{increase}} = 6$

1st index

ON bit: a, d

OFF bit: b, c

2 2

ab → single bit diff
ac → " " "
db → " " "
a c → " " "

b a
c a
b d
c d

\+

= 8

total increment = $\underbrace{2 \times 2}_{\text{pair}} \times \underbrace{2}_{\text{inver.}}$

```
public static long sumOfBitDifference(int[] arr){
    long res = 0;
    for(int i = 0; i < 32; i++) {
        long count0 = 0; // OFF
        long count1 = 0; // ON
        for(int val : arr) {
            // check if ith bit is ON in val or not
            int bm = (1 << i);
            if((val & bm) != 0) {
                // bit is ON
                count1++;
            } else {
                // bit is OFF
                count0++;
            }
        }
        count0 = arr.length - count1
        res += count0 * count1 * 2;
    }
    return res;
}
```

pair

inverse

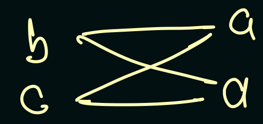
i=1 bm - 00010

i=0 bm = 00001

i ← loop

val

	4	3	2	1	0
a →	1	0	1	1	1
b →	1	1	0	0	1
c →	1	0	1	0	1
d →	1	1	0	1	0



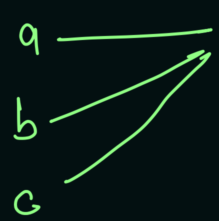
4 pair × 2 Inverse

= 8 inv

Count0 = ~~0~~ 1   ~~0~~ ~~1~~ 2

count1 = ~~0~~ ~~1~~ ~~2~~ 3   ~~0~~ ~~1~~ 2

res = ~~0~~ ~~6~~ ~~14~~ ~~22~~ 30



× 2 inv

1×3×2

Print Binary and Reverse Bits:

Input :- n = 91 decimal form → Binary form → 1 0 1 1 0 1 1

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

↓

Reverse

$(1\ 1\ 0\ 1\ 1\ 0\ 1)_2$

Output

Result → 1. 1 0 1 1 0 1 1

2. 109 →

$2^0 \to 1$
$2^1 \to 2$
$2^3 \to 8$
$2^4 \to 16$
$2^6 \to 64$

91

$2^0 \to 1$
$2^2 \to 4$ 5
$2^3 \to 8$
$2^5 \to 32$ 45 109
$2^6 \to 64 \to 64$

109

n = 0 0 0 - - 0 $\boxed{1}$ 0 1 1 0 1 1

Binary → 1 0 1 1 0 1 1

Rev → 1 1 0 1 1 0 1 = $(109)_{10}$

Check divisibility by 3 :→

n = 7 3 2 4 2 6 is it divisible by 3 or not

Sum of all digit = 24.

if it is divisible by 3 then whole no. is divisible by 3

Input → Binary String

is it divisible by 3, without using conversion

$n = (1\,0\,0\,1\,1\,1\,0\,1\,1)_2$ check divisibility by 3 Concept → Hamming Weight

└ in Binary $(11)_2$

if n is divisible by 11 in binary form

then it must be divisible by '3' in decimal form,

How to check divisibility of 11 .

| Sum of digit of odd Index − Sum of digit of even Index | % 11 == 0

then we can say than 'n' is divisible by 3.

Count Set Bits in first N - natural numbers :→

$n = 7$

n=1

0 -	0	0	0	0
1 -	0	0	0	1
2 -	0	0	1	0
3 -	0	0	1	1
4 -	0	1	0	0
5 -	0	1	0	1
6 -	0	1	1	0
7 -	0	1	1	1
8 -	1	0	0	0
9 -	1	0	0	1
10 -	1	0	1	0
11 -	1	0	1	1
12 -	1	1	0	0

$x = 3$

2^x

3rd bit

0-base index

$8/2 = 4$ bit are ON

Combination of x-bits.

1. Find nearly number which is of type 2^x from n.

0 -	0	0	0	→ 0
1 -	0	0	1	→ 1
2 -	0	1	0	→ 1
3 -	0	1	1	→ 2
4 -	1	0	0	→ 1
5 -	1	0	1	→ 2
6 -	1	1	0	→ 2
7 -	1	1	1	→ 3

12 set bits

i=0	1<<i	→ 1
1	1<<i	→ 2
2		→ 4
3		→ 8
4		→ 16

$\boxed{2^3 = x}$

$n - 2^x$

$\underbrace{\frac{2^x}{2} \times x}$

$$\text{Solve}(n) = \underbrace{2^{x-1} \times x}_{\text{first part}} + \underbrace{(n - 2^x + 1)}_{\text{second part}} + \underbrace{\text{Solve}(4)}_{\text{rec Rep}}$$

result n=12

$$\boxed{f(n) = 2^{x-1} \times x + (n - 2^x + 1) + f(n - 2^x)}$$

where x is power of 2, from a number which is smaller nearer from n.

Recursion

Smaller problem (n == 0) return 0;

```java
private static int powerOf2(int n) {
    int num = 1;
    int x = 0;
    while(num <= n) {
        num = (num << x);
        x++;
    }
    x--;
    return x;
}
```

shift by 1

$n = 12$

x	num
0	$1 \rightarrow 2^0$
1	$10 \rightarrow 2^1$
2	$100 \rightarrow 2^2$
$x = 3$	$1000 \rightarrow 2^3$
4	$10000 \rightarrow 2^4$

$x--$

16 is greater than 12

return x

Remaining part of Bit Manipulation →

1. Min Xor Pairs
2. Nth Palindromic Binary
3. Xor Queries Of A Subarray
4. Minimum Flips To Make A Or B Equal To C
5. Find Longest Awesome Substring

question of upcoming bit test