Space Complexity :-

- $O(n)$ → `int ar[N]`
- $O(n^2)$ → `int mat[N][N]`

Even if it is `int mat1[N][N], int mat2[N][N]`

→ $N^2 + N^2$

$= 2N^2$

$= O(N)$

★ Any online platform → space limit [256 MB or 128 MB]

↳ Errors → space limit exceeded.

. `int ar[N]` $1 \le N \le 10^4$ will this work?

worst case `ar[10^4]` → 4×10^4 B

for int → 40KB ✓

1 MB = 10^6

1KB = 10^3

. `int ar[N]` $1 \le N \le 10^6$

→ 4×10^6

→ 4MB ✓

. 10^7 → 40 MB ✓

. 10^8 → 400MB ✗

SC

. for int $N \le 10^6$ to 10^7

for bool $N \le 10^8$

- bool ar [10^8] → 1 x 10^8 B = 100MB ✓
  for bool

- int ar [10^3] [10^3] → 4×10^6 → 4MB ✓
- int ar [10^4] [10^4] → 4×10^8 → 400MB ✗
- bool ar [10^4] [10^4] → 1×10^8 → 100 MB ✓

2D array	→ $N \leq 10^3$	(int)
	→ $N \leq 10^4$	(bool)

Time limit	→ $N \leq 10^8$	
Space limit	→ $N \leq 10^{6-7}$	(int)
	$\leq 10^8$	(bool)

Problems :- $1 \leq ar[i] \leq 10^8$

	0	1	2	3	4	5	6	7	8
ar_N :	5	1	2	12	5	16	1	12	2

- Everything is repeated twice except one number

o/p : 16

Solutions

1) BF: TC : N^2 SC : 1 [count method]

2) EXOR TC : N SC : 1 [EX-OR Method]

Approach 1 :- count methods two loops

Approach 2 : $a \wedge a = 0$

$a \wedge b \wedge a = b$

: perform ex-or on all the elements

```
func findNonRepeated#(arr: [Int]) -> Int {
    var out = 0
    for element in arr {
        out = element ^ out
    }
    return out
}
```

TC : N

SC : 1

Problem 2 :

i/p	o/p
a b	ab
5 2	25
2 10	1024
3 4	81

1) BF : TC : b SC : 1

Approach 1 :- loop b times TC : b SC : 1

Approach 2:

- $3^{21} \rightarrow 3^{x1} \cdot 3^{x2} \cdot 3^{x3} \cdots$

$x1, x2, x3$ — Power of 2, unique

$\rightarrow 3^{16} \cdot 3^{4} \cdot 3^{1}$

- $3^{48} \rightarrow 3^{32} \cdot 3^{16}$

$3^{21} = 3^{16} \cdot 3^{4} \cdot 3^{1}$

$21 = 1\ 0\ 1\ 0\ 1$

# bit position	$x = x * x$	ans = 1
0	3	3
1	3^2	3
2	3^4	3^5
3	3^8	3^5
4	3^{16}	3^{21}
5	3^{32}	3^{21}
6	3^{64}	⋮
7	⋮	
⋮		3^{21}

$3^{48} = 3^{32}\ 3^{16}$

$= 1\ 1\ 0\ 0\ 0\ 0$

# bit position	$x = x * x$	ans = 1
0	3	1
1	3^2	1
2	3^4	1
3	3^8	1
4	3^{16}	3^{16}
5	3^{32}	3^{48}
6	3^{64}	
7		
⋮		
31	$3^{3^{31}}$	3^{48}

→

```
func compute (a: Int, b: Int) -> Int {
    var x = a
    var ans = 1
    for i in 0 ... 31 {
        (b>>i)&1 != 0
        if (CheckBit(b, i)) {
            ans = ans * x
        }
        x = x * x
```

```
}
return ans
```

→

```
var x = a
var ans = 1
while (b > 0) {
    if ((b&1) != 0) {
        ans = ans * x
    }
    x = x * x;
    b = b >> 1;
}
return ans
```

→ $b = b/2$ (right shift is halving)

$O(\log_2 b)$

$O(1)$

Prob: 5 12 5 12 **16** 14 5 14 12 14

00101 01100 00101 01100 10000 01110 00101 01110 01100 01110

Everything is repeated thrice

except one #, find it

o/p: 16

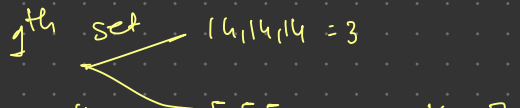
1 0 0 0 0

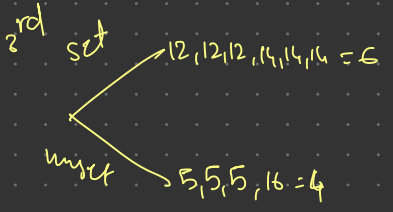
3rd 2nd 1st 0th

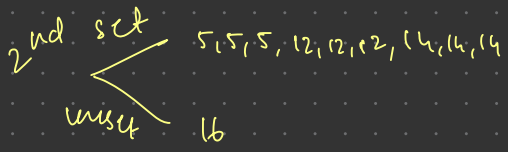
0^{th} set: 5,5,5, = 3

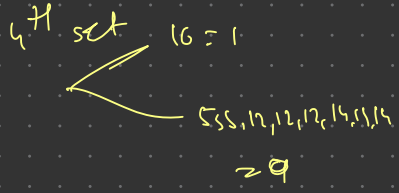
0^{th} unset: 12,12,12,14,14,14,16 = 7

1) BF: TC(N^2) SC(1)










```
func compute ( arr : [Int] ) → Int {
    var ans = 0
for i in 0 --- 31 {
    var count = 0
    for j in arr {
        if ( checkBit (j,i)) {
            count + = 1
        }
    }
    if (count % 3 == 1) {
        ans = ans | 1 << i
    }
    return ans
}
```

O(31N) TC

↳ O(N)

Problem:

0	1	2	3	4	5	6	7	8	9
5	3	2	5	1	6	4	7	8	4
0101	0011	0010	0101	0001	0110	0100	0111	1000	0100

N [1, N-2] Two numbers are repeating twice (different)

[1, 8] o/p : 4,5

BF

BF N^2, 1

```
func count ( arr : Int) → [Int] {
    var out = [Int]()
    for i in 0 ... arr.count-1 {
        for j in (i+1) ... arr.count-1 {
            if i == j {
                out.append (i)
                break
            }
        }
        if out.count == 2 { break }
    }
    return out
}
```

Approach 2:-

int x = 0

$(N) \times \overset{N-1}{\underset{i=0}{}}$ $x = x \wedge arr[i]$

[// ^ all elements 4 $s will be removed]

$(N-1) \oplus_{i=1}^{N-2} x = x \oplus i$

then $x = 4 \oplus 5$

[again ^ on 1 to 8 → all other [1, N-2] eliminated & 4's remain]

4 : 0 1 0 0

5 : 0 1 0 1

0 0 0 1 = 1

E-OR on two different #'s will be non-zero: check which bit is different in both the #s (any 1 bit no need for all bits that are diff)

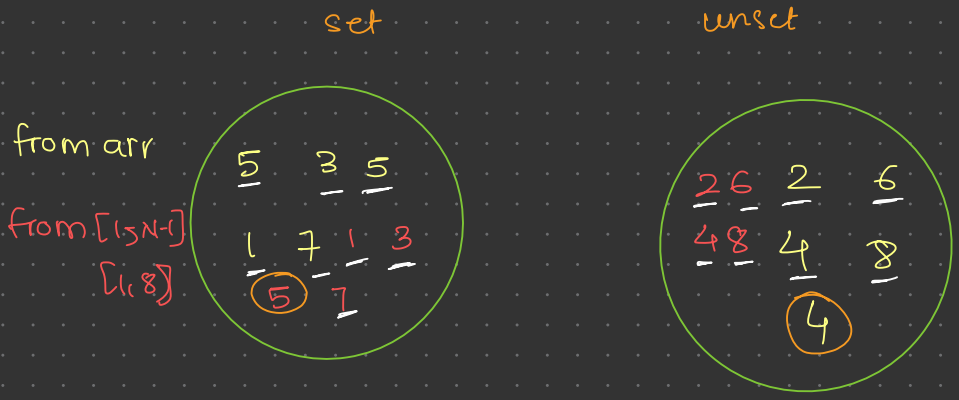
→

```
a = 0
while b != 0 {
  if (b & 1 == 0) {
      a += 1
  }
  b = b >> 1
}
return a
```

```
pos = 0
while (x > 0) {
  if (x & 1) != 0 {
    return pos
  }
  pos++
  x = x >> 1
}
```

TC = log x

we know 0th bit is set in this case



1 bit position

var a=0 , b=0

(N) → i=0 to N-1

if (set)
a ^= arr[i]
else
b ^= arr[i]

(N-1) → i=1 to N-2

if (set)
a ^= i
else
b ^= i

→ return a , b

TC : N + N-1 + log x + N + N-1

≈ 4N + log x → (N) , (1)

Problem:

arr : 5 2 12 −6 20 3 4

k = 10 → True { 5,2,3 } { 12,−6,4 }

= 100 → false

Subset who's sum = k

l = arr.count

for i in 0 ... l-1 {

---

5 2 −14

2^N for i in 0 ... ((1<<N)-1) {

c = 0, sum = 0

while(i != 0){

if((i&1) != 0) {

N sum += arr[c]

}

$2^N \cdot N$ } i >>= 1, c += 1

if sum == k {

return true

}

}

return false

N^2

```
for i in 0 --- N-1 {
    for j in 0 --- N-1 {
        ans ^ = (arr[i] + arr[j])
    }
}
```

→ optimise

5	12	7

~~5+5 ^ 5+12 ^ 5+7 ^~

~~12+5 ^ 12+12 ^ 12+7 ^~

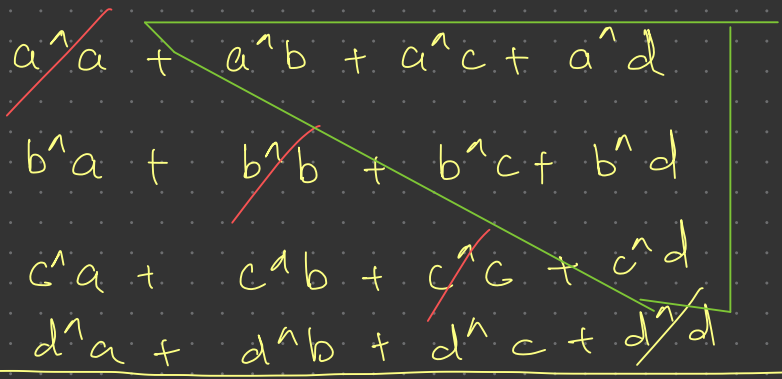
~~7+5 ^ 7+12 ^~ 7+7

N

```
for i in 0 --- N-1 {
    ans ^ = (arr[i] + arr[i])
}
```

```
int ans = 0
for i in 0 --- N-1 {
    for j in 0 --- N-1 {
        ans += (arr[i] ^ arr[j])
    }
}
```

a b c



$= 2\,(a \wedge b + a \wedge c + b \wedge c)$

```
for i in 0 --- N-1 {
    ans += arr[i] ^ a
```

5 7 12

5 ^ 12 7 ^ 7 12 ^ 7

0101 0101 1100

1100 0111 0111

1001 0010 1011

9 (8+1) + 2 + 11 (8+2+1)

$= 2\times2^0 + 2\times2^1 + 0\times2^2 + 2\times2^3$

$+ 0\times2^4 + 0\times2^5 + \cdots$

$= 22\times2 = 44$

5: 3210 0101

12: 1100

7: 0111

0th pos set

5,7 (2)

12 (1)

unset

$\Rightarrow 2\times1$

$\Rightarrow 2\times1\times2^0 = 2$

1st pos set

7 (1)

5,12 (2)

unset

$\Rightarrow 1 \times 2 \times 2^1 = 4$

2nd pos set → 5, 12, 7 (3)

$3 \times 0 \times 2^2 = 0$

unset → — (0)

3rd pos set → 12 (2)

$1 \times 2 \times 2^3 = 16$

unset → 5 7 (1)

$2 + 4 + 0 + 16$

$= 22$

$\Rightarrow 22 \times 2 = 44$

(reverse direction)

```
ans = 0
for i in 0 --- 31 {
    set = 0
    for j in 0 --- N-1 {
        if (checkbit (arr[j], i) {
            set += 1
        }
    }
    ans += ((set) * (N-set) * (1 << i))
}
return ans * 2.
```