Content

- Bitwise operators & properties.
- Check-Bit
- Unique Element
- Single Element - II
- Single Element - III

Bitwise Operators ⇒ &, |, ^, ~, <<, >>

Truth table.

same same, puppy shame.

a	b	a|b	a&b	a^b	~a
0	0	0	0	0	0
0	1	1	0	1	0
1	0	1	0	1	1
1	1	1	1	0	1

	(2^7) 7	(2^6) 6	(2^5) 5	(2^4) 4	(2^3) 3	(2^2) 2	(2^1) 1	(2^0) 0	
a = 29 :	0	0	0	1	1	1	0	1	
b = 18 :	0	0	0	1	0	0	1	0	
a&b :	0	0	0	1	0	0	0	0	= (16).
a|b :	0	0	0	1	1	1	1	1	= (31)
a^b :	0	0	0	0	1	1	1	1	= (15)

Properties

$a \& b = b \& a$

$a | b = b | a$

$a \wedge b = b \wedge a$

// Commutative Property.

$a \& b \& c = b \& a \& c = c \& a \& b$

$a | b | c = b | a | c = c | a | b$

$a \wedge b \wedge c = b \wedge a \wedge c = c \wedge a \wedge b$

// Associative Property

a = 10 :

```
  1 0 1 0
& 0 0 0 1
---------
  0 0 0 0
```

a = 11 :

```
  0 0 1 1
& 0 0 0 1
---------
  0 0 0 1
```

$a \& 1$ → $= 0$ {last bit of a is '0'} ⇒ No. is even

$a \& 1$ → $= 1$ {last bit of a is '1'} ⇒ No. is odd.

$a \& 0 = 0$

$a \& a = a$

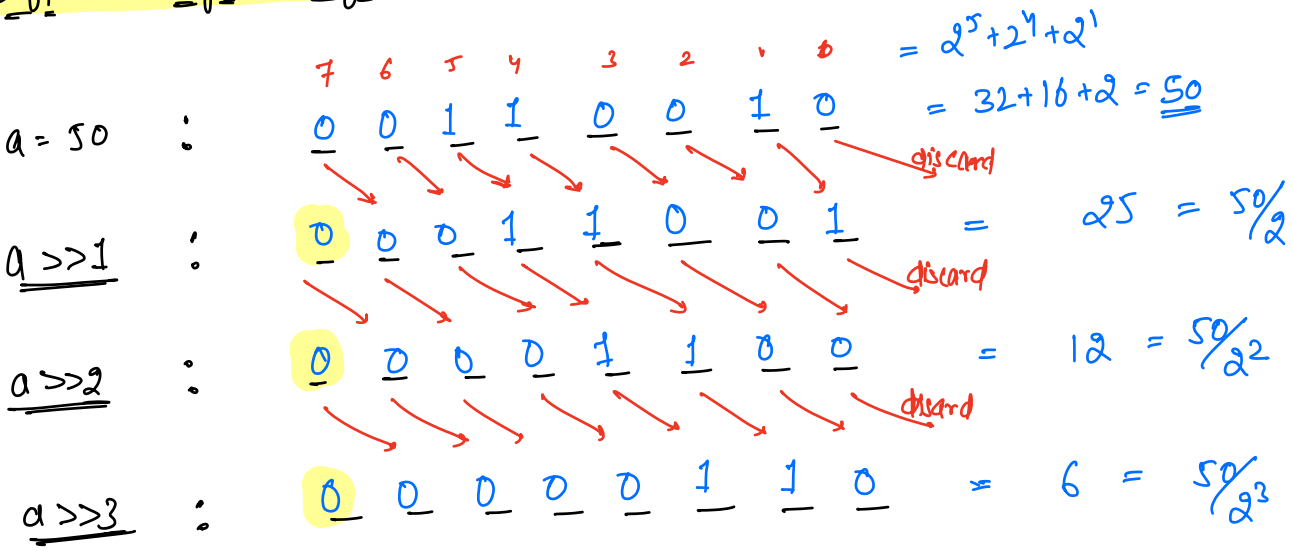
$a | 0 = a$

$a | a = a$

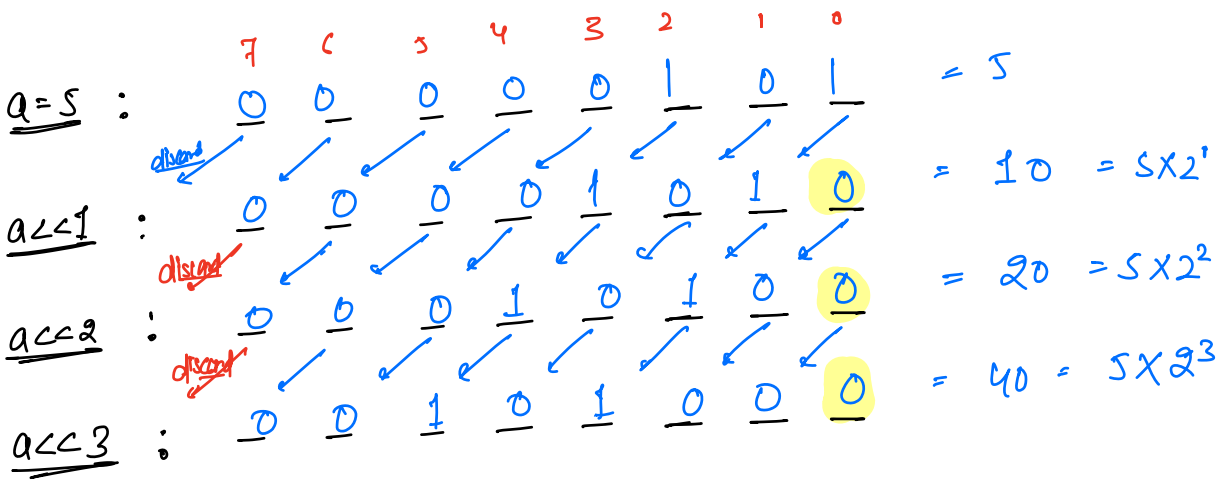
$a \wedge 0 = a$

$a \wedge a = 0$

Left & Right Shift →



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



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

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

Q: Given a number N & i. Check if ith-bit is set.

set → 1, unset → 0

N = 53 : 6 5 4 3 2 1 0 → 0 1 1 0 1 0 1

i = 2 → set.

i = 3 → unset.

N >> 1 → 0 0 1 1 0 1 0

N >> 2 → 0 0 0 1 1 0 1

i = 0 → (N & 1) == 1 { 0th bit is set }

i = 1 → ((N >> 1) & 1) == 1 { 1st bit is set }

i = 2 → ((N >> 2) & 1) == 1 { 2nd bit is set }

.

.

.

i → ((N >> i) & 1) == 1 { ith bit is set }

```
boolean checkBit ( N , i ) {
    if ( ((N >> i) & 1) == 1 ) {
        return true;
    }
    else {
        return false;
    }
}
```

T.C → O(1)
S.C → O(1)

{ To do → try with left shift operator }

Q: Given N elements, every element repeats twice except one. Find unique element.

arr[7]: { 3 2 3 7 2 8 7 } : ans = 8.

// ans = XOR of all the elements.

[T.C → O(N)
S.C → O(1)]

Pseudo-code:

```
val = 0
for( i = 0; i < N; i++) {
        val = val ^ arr[i]
}
return val
```

Q) Every element repeats thrice except one. Find unique element.

Constraints : $1 <= N <= 10^6$, $1 <= arr[i] <= 10^9$

arr[13]: { 5 7 5 4 7 11 11 9 11 7 5 4 4 }

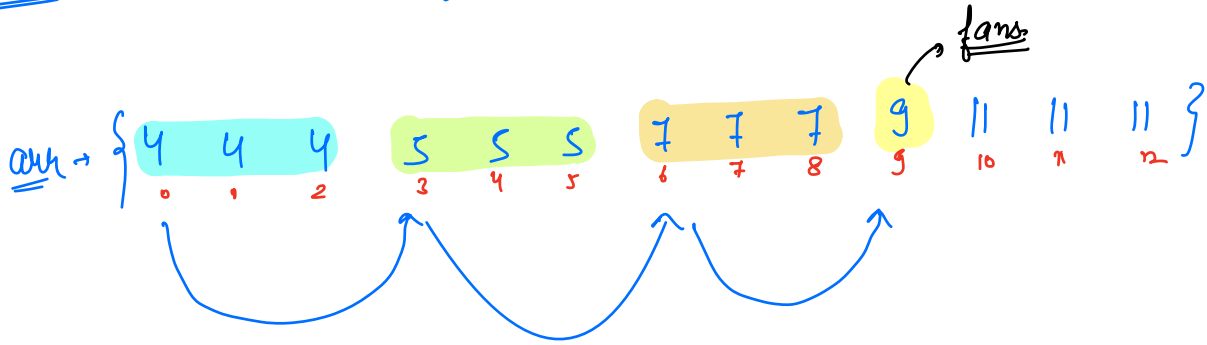
idea-1 for every element, iterate & find it's frequency.

T.C → $O(N^2)$, S.C → $O(1)$

idea-2. for every element, store its frequency in hashmap.

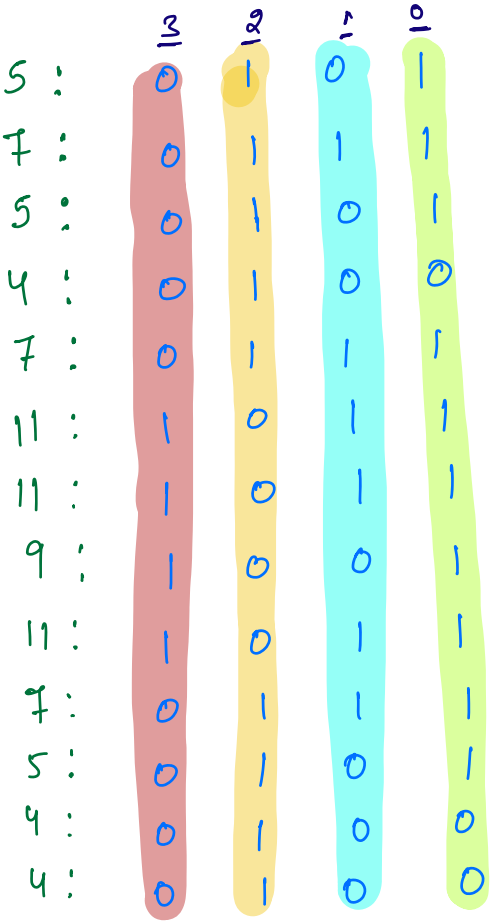
TC → $O(N)$, S.C → $O(N)$

idea-3 : Sort the array + { iterate & check every 3 elements }



T.C → $O(N \log N)$, S.C → $O(1)$

arr[13]: { 5 7 5 4 7 11 11 9 11 7 5 4 4 }
0 1 2 3 4 5 6 7 8 9 10 11 12



{ Count of set-bits at every position }

3rd	2nd	1st	0th
↓	↓	↓	↓
unique = { 1	0	0	1 }

ans → { 1 0 0 1 } = 9.

idea: At every bit position, iterate on array elements and find count of set-bits for that bit-position.

if (count % 3 == 0) { In unique element, ith bit will be unset }

else. { In unique element, ith bit will be set }

Pseudo-code:

```
ans = 0   [Initially all bits are '0']

for( i = 0 ; i < 32 ; i++ ){
        //find count of set bits at ith bit position
          of all the elements
        count = 0
        for( j = 0 ; j < N ; j++ ){
            if ( checkBit( arr[j], i )){
                count++
            }
        }
        if ( count % 3 == 1 ) { //extra bit.
            //set ith-bit in your 'ans'.
                ans = ans + 2^i
                ans = ans | (1<<i)      ] Revise & check.
}
return ans;
```

Eg:

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

no of iterations = 32·N

T.C → O(N)

S.C → O(1)

Some variations

① Every element repeats thrice except 1.

② Every element repeats thrice & one element is repeating twice.

[C%3 == 2]

③ Every element is repeating 5 times except 1.

[C%5 == 1]

④ Every element is repeating 5 times & one element is repeat twice.

[C%5 == 2]

⑤ Every element is repeating 5 times & one element is repeat thrice

[C%5 == 3].

Q) Given N elements every element repeats twice except 2 unique elements. Find two unique elements.

Eg: arr[6] → { 3 6 4 4 3 8 } unique elements = 6, 8

arr[4] → { 4 9 9 8 } unique elements = 4, 8

Idea-1. for every element, iterate and find the frequency.

T.C → $O(N^2)$, S.C → $O(1)$

opt

Idea-2. for every element, get frequency using hashmap.

T.C → $O(N)$, S.C → $O(N)$

Idea-3. Sort the array + iterate and get unique elements.

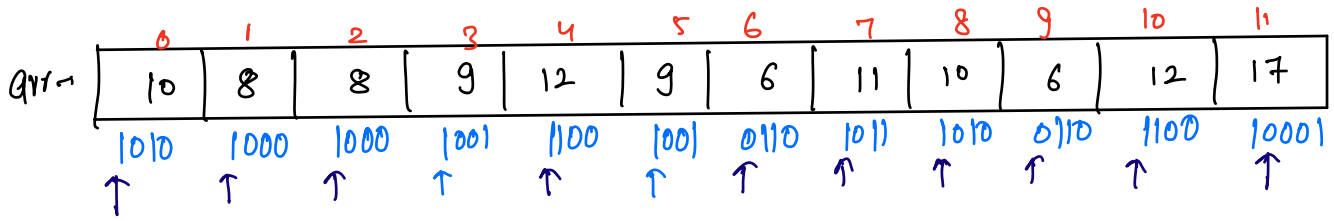
arr → { 3 3 4 4 6 8 }

T.C → $O(N \log N)$, S.C → $O(1)$

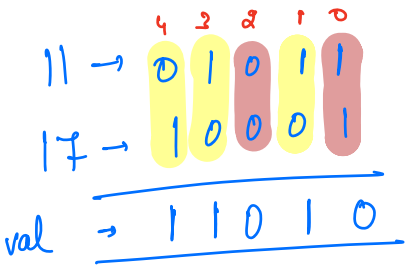
Idea4: XOR of all the elements.

{ 3 ^ 6 ^ 4 ^ 4 ^ 3 ^ 8 } = { 6 ^ 8 } = 14

[How to extract 2-unique elements from this value.]



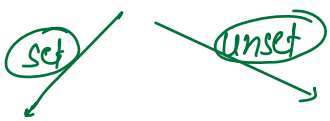
val = 11 ^ 17



observation: bits are different at pos → [1, 3, 4].
for 2 unique elements.

Split the array based on 1st bit position.

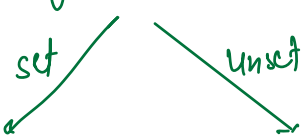
1st bit.



{ 10, 6, 11, 10 6 } — s1 — xor of s1 = 11

{ 8, 8, 9, 12 9, 12, 17 } — s2 — xor of s2 = 17

Split the array on 3rd bit position



{ 10 ^ 8 ^ 8 ^ 9 12 ^ 9 ^ 11 ^ 10 ^ 12 } — s1 ⇓ (11)

{ 6 ^ 6 ^ 17 } — s2. ⇓ (17)

pseudo-code

step 1 : Take XOR of all the elements

```
val = 0 , pos = -1
for( i = 0 ; i < N ; i++) {
{       val = val ^ arr[i]
}
```

N

step 2 : get a set-bit position in val

```
for( i = 0 ; i < 32 ; i++) {
{       if ( checkBit ( val, i)){
        {       pos = i
                break
        }
}
```

32

step 3 : Split the array elements on basis of pos^{th} bit

```
s1 = 0 ,  s2 = 0
for( i = 0 ; i < N ; i++) {
{       if ( checkBit ( arr[i] , pos)){
        {       s1 = s1 ^ arr[i]
        }
        else {
                s2 = s2 ^ arr[i]
        }
}
```

N

step 4 : print (s1 and s2)

T.C → O(N)
S.C → O(1)

Q: Given N array elements. Array contains all elements from [1 N+2] except 2 elements. find 2 missing elements.

[1, 6]

arr[4] : { 3 6 1 4 }

missing elements

[2, 5]

[1 7]

arr[5] : { 1 6 4 7 5 }

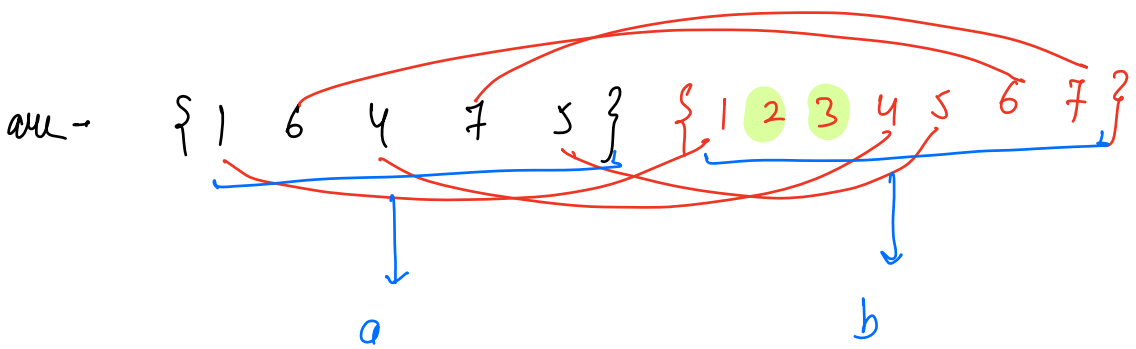
missing elements

[2, 3]

ideas. →

- → nested loop
- → hashmap
- → sort
- → Swap elements to keep them at their correct position

Todo



{ XOR of all elements from 1 to N+2 }

[a ^ b = XOR of all the elements]

```
b = 0
for( i = 1 ; i <= N+2 ; i++){
        b = b ^ i ;
}
a = 0
for( i = 0 ; i < N ; i++){
        a = a ^ arr[i]
}
val = a ^ b
```

b = 0

value of i	value of b
1	0^1 = 1
2	(1^2)
3	(1^2^3)
4	(1^2^3^4)
:	
i	
N+2	(1^2^3^4...(arr))

→ Remaining steps of previous question. {#todo}

{ T.C → O(N) S.C → O(1) }

→ Repeat & Missing Number. } → very similar to above question.