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

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

a^b : (1) when (a) is different from (b)

NOT (~) operator

a	~a
0	1
1	0

Properties of Bitwise operators

1) $a \wedge b = b \wedge a$
   $a \& b = b \& a$
   $a | b = b | a$
   } Commutative

2) $a \wedge b \wedge c = (a \wedge b) \wedge c$
$= a \wedge (b \wedge c)$
} Associative (&, |)

3) $a \wedge 0 = a$

$1 \wedge 0 = 1$

$0 \wedge 0 = 0$

4) $a \wedge a = 0$

5) Left Shift (<<)

Quiz $a = 15$

$a << 2 = a \times 2^2$

$= 15 \times 2^2 = 60.$

$a << N = a \times 2^N$ (assuming no overflow)

Quiz $2^N = K << N$

↳ O(1)

pow(a,n) ⇒ $O(\log_2 N)$

6) Right Shift (>>)

Quiz $a = 29$

$a >> 2 \Rightarrow \frac{a}{2^2} = \frac{29}{2^2} = \frac{29}{4}$

⇒ 7

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

(No overflow)

a = 40
↓ >>1
20
↓ >>1
10
↓ >>1
5
↓ >>1
2
↓ >>1
1 →(>>1) 0 →(>>1) 0

Q Given a number (N), check if i^{th} bit is set / unset.

```
          6 5 4 3 2 1 0        (i = 2)
N ⇒       0 1 1 0 1 0 1  ← N
          0 0 0 0 1 0 0  ← (1<<2)
        &
  ---------------------
          0 0 0 0 1 0 0
```

```
bool CheckBit(N, i) {
    if (N & (1<<i)) {
        return true;
    }
    return false;
}
```

⇒ TC: O(1)

```
bool CheckBit(N, i) {
    if((N>>i)&1) {
        return true;
    }
    return false;
}
```

⇒ O(1)

Q.1 Amazon Flipkart etc

Given an array of +ve numbers, where each element appears twice except one element which appears once. Find the no. which appears once.

$a \wedge a = 0$

A : {1, 2, 4, 2, 4, 3, 1}

$1 \wedge 2 \wedge 4 \wedge 2 \wedge 4 \wedge 3 \wedge 1$ ⇒ (3)

⇒ XOR

TC : O(N)

SC : O(1)

Q. Given an array of +ve integers, all no's appears twice except two numbers which are appearing once. Return these 2 single numbers.

Google
Amazon

A: {3, 6, 4, 4, 6, 8}

⇒ 3, 8.

A: {4, 9, 9, 8} ⇒ 4, 8

A: {1, 2} ⇒ 1, 2

Ex :- {a, b, c, d, a, b, d, e}

$a \wedge a = 0$

$a \wedge 0 = a$

xor = a ^ b ^ c ^ d ^ a ^ b ^ d ^ e

xor = c ^ e

Approach #1

Frequency Hashmap

TC : O(N)

SC : O(N)

↳ map

#2: Sorting

{ 3, 6, 4, 4, 6, 8 }

↓ Sort

{ 3, 4, 4, 6, 6, 8 }

TC : O(NlogN)

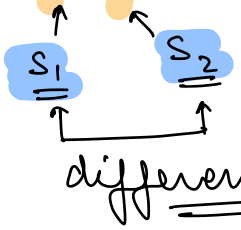
SC : Depends on Sorting Algo.

#3

{ a, b, c, d, a, b, d, e }

xor = a ^ b ^ c ^ d ^ a ^ b ^ d ^ e

xor = c ^ e



different no's

Can $S_1 \wedge S_2$ be zero ?

↳ NO

$S_1 \wedge S_2 > 0$ ⇒ It will have atleast one set bit.

A: { 3, 6, 4, 4, 3, 8 }

xor = 6^8 ⇒ 0 1 1 0
>0 ^ 1 0 0 0
6^8 ⇒ 1 1 1 0

Ex :- A: { 2, 1, 3, 3, 12, 2 }

xor ⇒ 1^12 = 3 2 1 0
0 0 0 1
^ 1 1 0 0
1 1 0 1

⇒ If x^{th} Bit is set in a^b the a and b will have different bits at x^{th} position.

Ex A: { 10, 8, 8, 9, 12, 9, 6, 11, 10, 6, 12, 17 }
1010 1000 1100 0110 1010 1100
1000 1001 1001 1011 0110 10001

XOR of A: 11^17
S_1 S_2
4 3 2 1 0
0 1 0 1 1 S_1
1 0 0 0 1 S_2
^
xor: 1 1 0 1 0

Group1 ⇒ 1st Bit is set
Group2 ⇒ 1st Bit is unset

POS = 1

Group 1 (Set)

10, 6, 11, 10, 6

XOR

10^6^11^10^6

= 11

S_1

Group 2 (Uset)

8, 8, 9, 12, 9,

12, 17

XOR

17

S_2

POS = 3

Group 1 (Set)

10, 8, 8, 9, 9,

12, 12, 10, 11

XOR

11

Group 2 (Uset)

6, 6, 17

17

Steps :-

1) Take XOR of complete Array (xor) $\Rightarrow$ O(N) ($s_1 \wedge s_2$)

2) Find any bit position which is set in xor $\rightarrow \log_2(Max)$

3) Based on this Bit position, segregate the array elements into two group. $\Rightarrow$ O(N)

4) Take XOR of group1 $\Rightarrow S_1$

   Take XOR of group2 $\Rightarrow S_2$

TC : $O(N + \log_2(Max))$ $32+N$

$\Rightarrow$ xor = a ^ b

pos = x // x^{th} Bit is set in xor

```
ans1 = 0
ans2 = 0
for(i = 0; i < N; i++){
    if(CheckBit(A[i], pos)){
        ans1 ^= A[i]
    }
    else
        ans2 ^= A[i]
}
return ans1, ans2;
```

Q. Given an Array of size N containing all elements from (1) to (N+2) except (2) elements. Find (2) missing no's.

Google Sharechat

* Array can't be modified.
* SC ⇒ O(1)

Ex: A = {1, 3, 6, 4} N = 4
⇒ 2, 5 [1, 6]

Ex:- {1, 6, 4, 7, 5} N = 5
[1, 7]
2, 3

Hint :-
A = {1, 3, 6, 4} N = 4
⇒ 2, 5 [1, 6]

[1, 2, 3, 4, 5, 6]
1 3 6 4

xor = 2 ^ 5

↳ same as last problem

1) Take XOR of all array elements

xor = 1 ^ 3 ^ 6 ^ 4

2) loop through numbers from 1 to N+2

xor = 1^3^6^4^1^2^3^4^5^6

xor = 2^5 = 0010
0101
0111

3) Find any set Bit position (POS) in xor.

4) i) Group the numbers from (1) to (N+2) based on the bit at position (POS) is set | unset.

0001 0010 0011 0100 0101 0110
1 2 3 4 5 6

0th Bit is set

Group (1)

1, 3, (5)
3, 1

XOR

1^3^5^1^3
⇒ (5)

Group (2)

(2), 4, 6
4, 6

XOR

(2)

ii) Group the Array elements based on bit position (POS) is set | Unset.

5) Take XOR of group (1) & (2) separately and find the (2) missing no's.

```
xor = 0
```

① `for(i=0; i<N; i++){` ⇒ O(N)

```
    xor ^= A[i]
}
```

② `for(i=1; i<=N+2; i++){` ⇒ O(N)

```
    xor ^= i;
}
```

③ // find set Bit position in xor ⇒ pos

↳ O($\log_2$(Max))

④⑤

```
ans1 = 0
ans2 = 0
```

i) `for(i=1; i<=N+2; i++){` ⇒ O(N)

```
    if(checkBit(i, pos)){
        ans1 ^= i;
    }
    else
        ans2 ^= i;
}
```

ii) `for(i=0; i<N; i++){` ⇒ O(N)

```
    if(checkBit(A[i], pos)){
        ans1 ^= A[i]
    }
    else
        ans2 ^= A[i]
}
```

print (ans1, ans2)

TC: $O(N + \log_2 Max)$

SC: $O(1)$

Q.3 Google Given an Array, every element appears thrice except (1) element, which appears once. Find the single no. [* SC : O(1)]

A: {5, 7, 5, 4, 7, 11, 11, 9, 11, 7, 5, 4, 4}

(9)

A: {5, 7, 5, 4, 7, 11, 11, 9, 11, 7, 5, 4, 4}

5 ⇒ 0 1 0 1.
7 ⇒ 0 1 1 1.
5 ⇒ 0 1 0 1.
4 ⇒ 0 1 0 0
7 ⇒ 0 1 1 1.
11 ⇒ 1 0 1 1.
11 ⇒ 1 0 1 1.
9 ⇒ 1 0 0 1.
11 ⇒ 1 0 1 1.
7 ⇒ 0 1 1 1.

In how many no's 0th Bit is Set = 3 + 3 + 3 + 1

= 10

⇒ 10 % 3 != 0, 0th Bit is Set in ans.

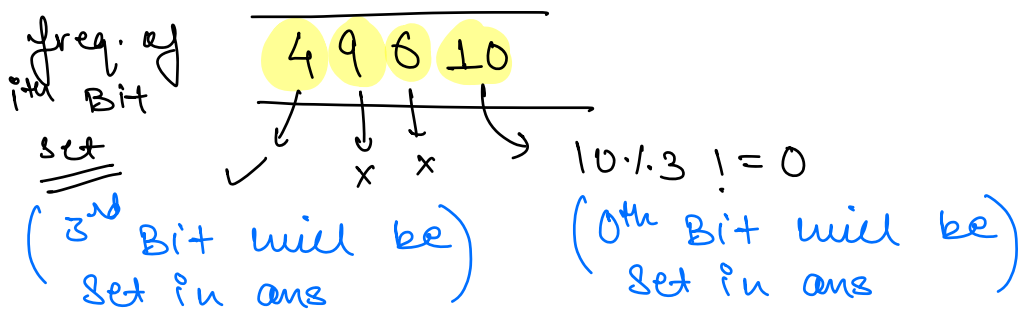
In how many no's 1st Bit is Set = 3 + 3

= (6)

5 ⇒ 0 1 0 1 ⇒ 1st Bit is not set in ans

4 ⇒ 0 1 0 0

4 ⇒ 0 1 0 0



ans = 1 0 0 1

Approach

1) Iterate through all the Bit positions :-

for ith Bit position :

Iterate on all the array elements & check :

if ith Bit is set : count++

2) If count % 3 != 0 for ith Bit :

ith Bit is set in ans.

3) If count % 3 == 0 for ith Bit :

ith Bit is not set in ans.

Java | C++ (Int)

```
ans = 0
for ( i = 0; i < 32 ; i++ ) {
    // count the no. of set bits at
    // ith positions
    count = 0;
    for ( j = 0; j < N; j++ ) {
        if ( checkBit ( A[j], i ) )
            count++ ;
    }
    if( count % 3 != 0 ) {
        ans = ans | (1<<i)
    }
}
return ans.
```

TC: $O(\log_2(Max) * N)$

SC: $O(1)$

Doubts

ans = O_3 2 1 0 (set)

= 0 0 0 0

0 0 1 0

OR

0 0 1 0

32 ?

1 to 1000

$\log_2$(Max)

$\log_2$(1000) ⇒ 9

3x3 ?

1 to 2^{48} ⇒ 48

$\log_2 2^{48} = 48$

3x3

	0	1	2
0	1	2	8
1	8	8	8
2	8	8	8

8

8

0,2

1,1

2,0