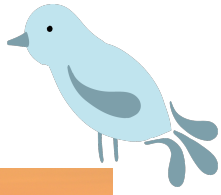
Bit Manipulation



Good Evening

Today's content

- → Bitwise operators & properties
- → check if bit is set or unset
- → Single element-I
- → Single Element - II
- → Single Element - III
- → Max AND Pair

Bitwise operators → &, |, ^, ~, <<, >>

A	B	A&B	A|B	A^B	~A	~B
0	1	0	1	1	1	0
0	0	0	0	0	1	1
1	1	1	1	0	0	0
1	0	0	1	1	0	1

1 → set
0 → unset

A & B → Both A & B must be 1, to get ans = 1

A | B → One must be 1, to get ans = 1

A ^ B → same same puppy shame

Bitwise operations

A = 20

B = 45

A = 0 0 0 1 0 1 0 0

B = 0 0 1 0 1 1 0 1

7 6 5 4 3 2 1 0

print (a&b) = 0 0 0 0 0 1 0 0 = 4

print (a|b) = 0 0 1 1 1 1 0 1 = 61

print (a^b) = 0 0 1 1 1 0 0 1 = 57

Properties

Commutative

$a \& b = b \& a$

$a | b = b | a$

$a \wedge b = b \wedge a$

Associative

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

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

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

* $a \& 1$
  - 0 (0th bit is unset)
  - 1 (0th bit set)

$a = 10 \& 1$

```
  a = 1 0 1 0
& 1 = 0 0 0 1
-------------
    0 0 0 0
```

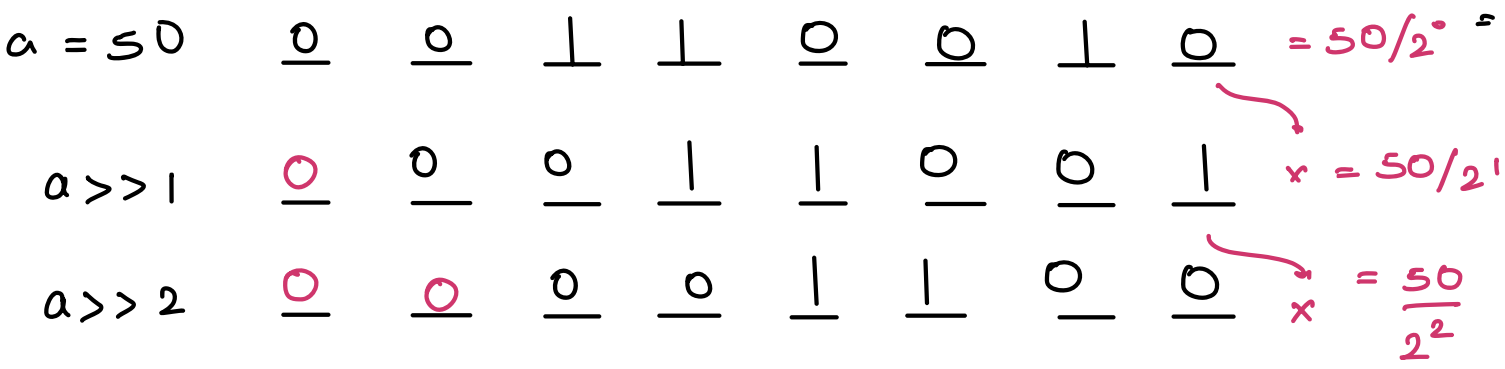
$a = 11 \& 1$

```
  a = 1 0 1 1
& 1 = 0 0 0 1
-------------
    0 0 0 1
```

$a \& 0 = 0$	$a \| 0 = a$	$a \wedge 0 = a$
$a \& a = a$	$a \| a = a$	$a \wedge a = 0$

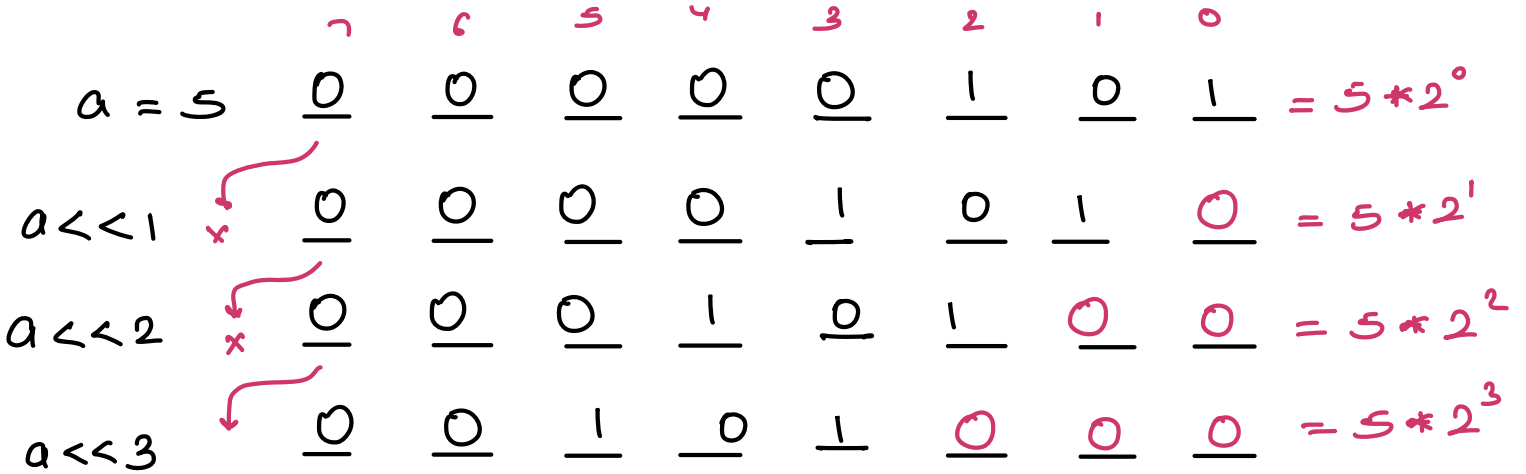
Left & Right shift operator

Right shift operator



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

Left shift operator



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

Q1→ Given a number N and i. Check if i^{th} bit is set.

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

i = 3 → false

i = 4 → true

i = 1 → false

i = 0

i = 0 N & 1 == 1 → 0^{th} bit is set

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

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

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

TC: O(1)

SC: O(1)

02. Given N elements, every element repeats twice except one. Find the unique element.

arr [7] = { 3 2 3 7 2 8 7 }
0 1 2 3 4 5 6

01. Xor of all the values in array

val = 3 ^ 2 ^ 3 ^ 7 ^ 2 ^ 8 ^ 7

val = 3 ^ 3 ^ 7 ^ 7 ^ 2 ^ 2 ^ 8 = 8
0 0 0 8

```
int val=0

for ( i=0 ; i<n; i++){
|   val= val ^ ar[i]:
3

return val:
```

03. Every element repeats thrice except one. Find unique element.

coming for one time

Constraints : $1 \le N \le 10^6$, $1 \le arr[i] \le 10^9$

arr[13] =	5	7	5	4	7	11	11	9	11	7	5	4	4

Ans = 9

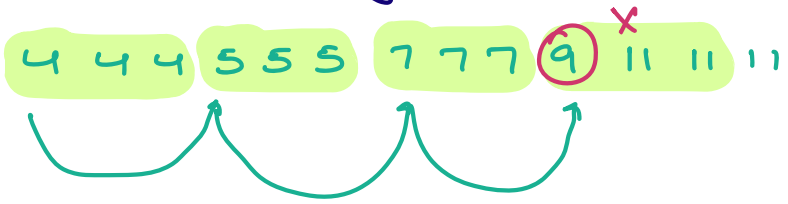
Brute force → For each ele, go & count the no. of times it is appearing

TC = $O(n^2)$
SC = $O(1)$

Idea 2 → → store frequency of arr ele in hashmap
→ Iterate & return the ele with freq = 1

TC = $O(n)$
SC = $O(n)$

Idea 3 → Sort the array



TC: O(nlogn)
SC: O(1)

Idea 4

arr[13]:

5	7	5	4	7	11	11	9	11	7	5	4	4

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

0^{th} bit sum = 10

1^{st} bit sum = 6

2^{nd} bit sum = 9

3^{rd} bit sum = 4

4 9 6 10 → Multiple of 3 since every ele is occuring thrice

4%3=1 9%3=0 6%3=0 10%3=1

1 0 0 1 = 9

Idea 4 → At every bit, calculate the count of no. of set bits in the arr

Say for i^{th} bit, count = c

if (c % 3 != 0) ans → set this particular bit.

```
int singlenumber2 (int [] ar, int n)
{
    int ans=0

    for (i=0; i<32; i++){
        // for ith bit, calculate the no. of set bits in
           arr elo.
        int c=0
        for (j=0; j<n; j++){
            if (checkbit ( ar[j], i) == true)
                c++;
        }
        if ( c % 3 != 0){
            ans = ans | (1<<i);        // ans + 2^i
        }
    }
}
```

TC : O(n)

SC : O(1)

* **Extentions**

01. Every ele is repeating thrice, except one ele which is repeating twice

Ans → The same code will work

02. Every ele is repeating 4 times, except one ele which is repeating once

Eg :- a ^ b ^ c ^ b ^ a ^ b ^ a ^ b ^ a

Ans → XOR of all elements

* One ele is occurring twice

Ans → if (c % 4 != 0)

* Except one ele which is repeating thric

Ans → XOR of all elements

10:19 → 10:26 pm

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

Eg:- arr[6] = { 3 6 4 4 3 8 } Ans = 6, 8

arr[4] = { 4 9 9 8 } Ans = 4, 8

Brute force → For each ele, go & count the no. of times it is appearing

TC = $O(n^2)$
SC = $O(1)$

Idea 2 → → store frequency of arr ele in hashmap

→ Iterate & return the ele with freq = 1

TC = $O(n)$
SC = $O(n)$

Idea 3 → Sort the array

TC: $O(n \log n)$
SC: $O(1)$

3 3 4 4 6 8

Idea 4 XOR of all elements

arr[6] = { 3 6 4 4 3 8 }

xor = 3 ^ 3 ^ 4 ^ 4 ^ 6 ^ 8 = 6 ^ 8

Obs = xor of complete array = xor of two distinct ele

arr[12] =

10	8	8	9	12	9	6	11	10	6	12	17
1010	1000	1000	1001	1100	1001	0110	1011	1010	0110	1100	10001

01. Step 1 → XOR of complete array

11 ^ 17

	4	3	2	1	0
11 =	0	1	0	1	1
17 =	1	0	0	0	1
	1	1	0	1	0

At this position both the distinct ele has diff bits.

* Idea → Split the array on the basis of 1st index bit.

arr[12] =

10	8	8	9	12	9	6	11	10	6	12	17
1010	1000	1000	1001	1100	1001	0110	1011	1010	0110	1100	10001

Splitting (1st idx posn)

0 → { 8, 8, 9, 12, 9, 12, 17 }

xor of this set = 17

1 → { 10, 6, 11, 10, 6, }

xor of this set = 11

* Split the array at 3rd index

arr[12] =

10	8	8	9	12	9	6	11	10	6	12	17
1010	1000	1000	1001	1100	1001	0110	1011	1010	0110	1100	10001

Splitting (on 3rd idx)

0 → { 6, 6, 17 }

xor of set = 17

1 → { 10, 8, 8, 9, 12, 9, 11, 10, 12 }

xor of set = 11

* arr[] = { 4, 7, 3, 3, 6, 6 }

Xor = 4 ^ 7

3 = 0 1 1

6 = 1 1 0

	2	1	0
4 =	1	0	0
7 =	1	1	1
v =	0	1	1

Splitting (0th idx)

0 → 4, 6, 6 — xor of set = 4

1 → 7, 3, 3 — xor of set = 7

* ——— unique III (int [] ar)

```
01. xor of complete arr
    int v = 0
    for (i=0; i<n; i++){
        v = v ^ ar[i];
    }
// v = xor of two distinct no.
02 Set bit in v
    posn = -1
    for (i=0; i<32; i++){
        if (checkbit(v, i) == true){
            posn = i
            break;
        }
    }
03. Split the arr based on this posn
    set = 0 , unset = 0
    for (i=0; i<n; i++)
        if (checkbit(ar[i], posn) == true){
            set = set ^ ar[i];
        }
        else
            unset = unset ^ ar[i];
    print(set); print(unset);
}
```

Homework

* Given n arr elements, Array contains all elements from [1 n+2] except 2 elements. Find those 2 missing ele.

04. Given N arr elements. Choose two indices (i, j) such that (i ≠ j) and arr[i] & arr[j] is maximum

↓ bitwise AND

ar[3] = {27 18 20}

what if (i == j) ⟶ max ele & max ele = max ele

* ar[3] = {27 18 20}

27 & 18 = 10010 = 18

27 & 20 = 10000 = 16

18 & 20 = 10000 = 16

27 → 11011

18 → 10010

20 → 10100

* Brute force → Consider all the valid pairs (i, j) & find the maximum of all pairs AND

TC: O(n²)

SC: O(1)

arr[4] = | 21 | 18 | 24 | 17 |

	4	3	2	1	0
21 →	1	0	1	0	1
18 →	1	0	0	1	0
24 →	1	1	0	0	0
17 →	1	0	0	0	1

21 & 18 = 16

21 & 24 = 16

21 & 17 = 17

18 & 24 = 16

18 & 17 = 16

24 & 17 = 16

* Optimised

Eg: arr[7] = | 26 | 13 | 23 | 28 | 27 | 7 | 25 |

	4	3	2	1	0
26 :	1	1	0	1	0
13 :	0	1	1	0	1
23 :	1	0	1	1	1
28 :	1	1	1	0	0
27 :	1	1	0	1	1
7 :	0	0	1	1	1
25 :	1	1	0	0	1

count of set bits	5	4	1	2	1 →
			Pair Not possible		Pair Not possible

Ans → 1 1 0 1 0

* Pseudocode

```
ans = 0

for ( i=31 ; i≥0 ; i--) {                    TC : O(n)
    // count no. of set bits at ith idx      SC : O(1)
    c=0
    for( j=0; j<n; j++){
        if ( checkbit (ar[j] , i) == true){
            c++;
        }
    }
    if ( c ≥ 2)
        ans = ans | (1<<i);        // ans = ans + 2^i
        for ( j=0 ; j<n; j++) {
            if ( checkbit (ar[j], i) == false) {
                ar[j] = 0;
            }
        }
}
```

* Maximise bitwise AND of triplets

```
if ( c ≥ 3) ans = ans + 2^i
```

distinct

* Calculate the count of pairs you can make for which bitwise AND is maximum (Google)

→ Follow the above approach

→ Traverse on the array & count the ele > 0

count = x

Ans = $x C_2$ or $\frac{x(x-1)}{2}$