Todays Content:

a) Number System Basics
b) Binary to Decimal & viceversa
c) Adding 2 Binary numbers
d) Bit Wise operations
    i) Basic Properties
    ii) Basic Problems

Decimal Number System → Each Digit: [0 1 2 3 4 5 6 7 8 9]

→ Each power: [10]

$10^3\ 10^2\ 10^1\ 10^0$

3 4 2 = 300 + 40 + 2 = $3*10^2 + 4*10^1 + 2*10^0$

2 5 6 3 = 2000 + 500 + 60 + 3 = $2*10^3 + 5*10^2 + 6*10^1 + 3*10^0$

Binary Number System → Each Digit: [0 1]

→ Each power: [2]

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

1 0 0 1 0 1 ⇒ $2^5 + 0 + 0 + 2^2 + 0 + 2^0 = 32 + 4 + 1 = 37$

$2^4\ 2^3\ 2^2\ 2^1\ 2^0$

1 0 0 1 1 ⇒ $2^4 + 0 + 0 + 2^1 + 2^0 = 19$

$2^4\ 2^3\ 2^2\ 2^1\ 2^0$

1 1 0 0 1 ⇒ $2^4 + 2^3 + 0 + 0 + 2^0 = 16 + 8 + 1 = 25$

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

0 0 1 0 1 0 ⇒ $2^3 + 2^1 = 8 + 2 = 10$

Note: Adding 0 at start won't change value

Decimal to Binary

2 | 37	- 1
2 | 18	- 0
2 | 9	- 1
2 | 4	- 0
2 | 2	- 0
2 | 1	- 1
0	

Rem

= 1 0 0 1 0 1

2 | 25	1
2 | 12	0
2 | 6	0
2 | 3	1
2 | 1	1
0	

Rem

= 1 1 0 0 1

2 | 27 :	1
2 | 13 :	1
2 | 6 :	0
2 | 3 :	1
2 | 1 :	1
0	

Rem

= $2^4\ 2^3\ 2^2\ 2^1\ 2^0$ = 1 1 0 1 1 = $2^4 + 2^3 + 2^1 + 2^0 = 27$

Note: Keep diving by 2 until Quotient becomes 0

Add 2 decimal numbers: $d = s\%10$ $c = s/10$: 10 decimal number system

Ex1:

c =		1	1	
	7	8	9	]
	1	4	2	]
	0			
s =	9	13	11	
d =	0	9	3	1

Ex2: 17/10 8/10 17/10

c:		1	0	1	
11/10	7	8	3	9	
	3	9	4	8	
s =	11%10	17%10	8%10	17%10	
d =	1	1	7	8	7

Add 2 Binary Numbers $d = s\%2$ $c = s/2$: 2 → Binary numbers

c =	1/2	3/2	2/2	1/2	
	0	1	1	0	
	1	0	1	1	0
1/2		0	0	1	1	1
s =	1%2	1%2	3%2	2%2	1%2
d =	0	1	1	1	0	1

c =	2/2	1/2	2/2	1/2	
	1	0	1	0	
	1	1	0	1	1
3/2	1	1	0	1	0
s =	3%2	2%2	1%2	2%2	1%2
d =	1	1	0	1	0	1

Why Binary?

In Electronics current passes

If voltage

- > = Certain limit : 1 (TODO)
- < Certain limit : 0

int n = 25;

In System data stored in Binary

n = 11001

Decimal to Binary

Binary to Decimal

print(n) = 25

These internal conversions are internally done by your System.

We don't have to worry.

Bitwise operations : { AND, OR, XOR, Invert, left shift, right shift }

`&` `|` `^` `~` `<<` At End `>>`

A	B	Both 1 A & B	one 1 A | B	diff 1 A ^ B	~A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

: Same Same puppy Shame

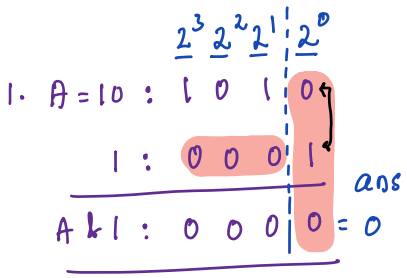
// a = 29 b = 19

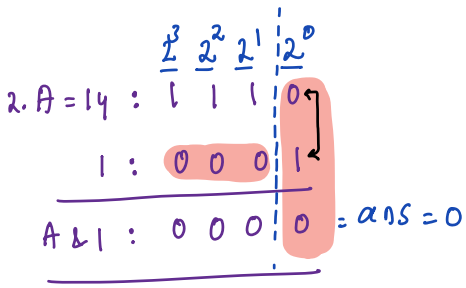
	2^4	2^3	2^2	2^1	2^0	Decimal
a :	1	1	1	0	1	
b :	1	0	0	1	1	
print(a^b):	0	1	1	1	0	→ $2^3+2^2+2^1 = 14$
print(a&b):	1	0	0	0	1	→ $2^4+2^0 = 16+1 = 17$
print(a|b):	1	1	1	1	1	→ $2^4+2^3+2^2+2^1+2^0 = 31$

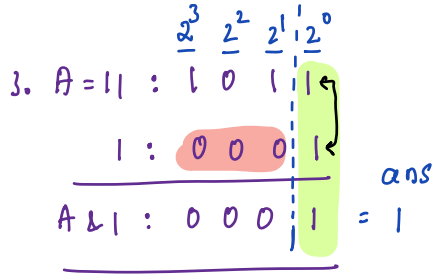
// a = 13, b = 10

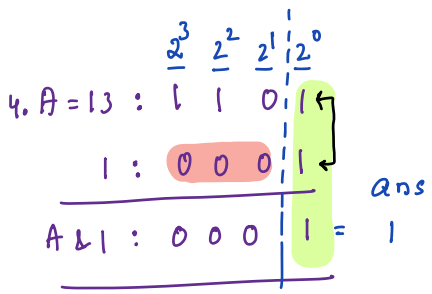
	2^3	2^2	2^1	2^0	Decimal
a :	1	1	0	1	
b :	1	0	1	0	
print(a&b):	1	0	0	0	→ $2^3 = 8$
print(a|b):	1	1	1	1	→ $2^3+2^2+2^1+2^0 = 15$.
print(a^b):	0	1	1	1	→ $2^2+2^1+2^0 = 7$.

Bit Wise Properties





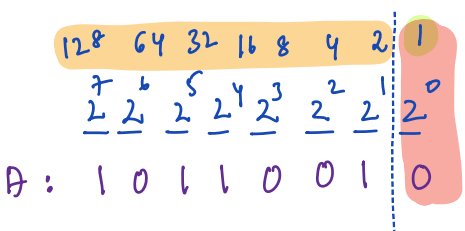


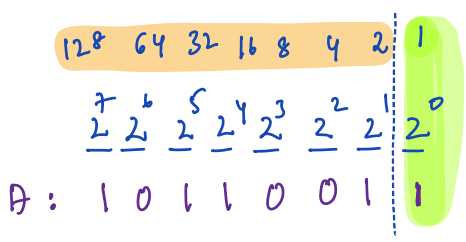


Obs: A & 1

- → 0 : In A, 0^{th} bit = 0 ⇒ A is even number
- → 1 : In A, 0^{th} bit = 1 ⇒ A is odd number

Ex: Q6 & Q7





Few More Interesting Properties

1. A & 1 → 0: A is even 1: A is odd

2. A & 0 → A: 1011 : 0
   0: 0000
   A & 0 : 0000 → 0

3. A & A → A: 1010 : A
   A: 1010
   A & A : 1010 = A

4. A | 0 → A: 1010 : A
   0: 0000
   A | 0 : 1010 = A

5. A | A → A: 1010 : A
   A: 1010
   A | A : 1010

6. A | 1 → TODO
   - A: if A is odd
   - A+1: if A is even

7. A ^ 0 → A: 1010 : A
   0 : 0000 1: Same: 0 Diff: 1
   A ^ 0 : 1010

8. A ^ A → A: 1010 : 0
   A: 1010 1: Same: 0 Diff: 1
   A ^ A: 0000 : 0

9. A ^ 1 → TODO
   - A+1: if A is even
   - A-1 : if A is odd

Commutative Property :

True/False

1. A & B = B & A = True
2. A | B = B | A = True
3. A ^ B = B ^ A = True
4. A & B & C = B & C & A = C & B & A = order won't matter
5. A | B | C = B | C | A = C | B | A = order won't matter
6. A ^ B ^ C = B ^ C ^ A = C ^ B ^ A = order won't matter

XOR Calculations

1. a^b^a^d^b = a^a^b^b^d = 0^0^d = 0^d = d

2. d^e^f^a^a^d^f = d^d^a^a^f^f^e = e

3. 1^3^5^3^2^1^5 = 1^1^3^3^5^5^2 = 2

1 : 0 0 1

3 : 0 1 1

1^3 : 0 1 0

5 : 1 0 1

1^3^5 : 1 1 1

3 : 0 1 1

1^3^5^3 : 1 0 0

2 : 0 1 0

1^3^5^3^2 : 1 1 0

1 : 0 0 1

1^3^5^3^2^1 : 1 1 1

5 : 1 0 1

2^2 2^1 2^0

1^3^5^3^2^1^5 : 0 1 0 = 2

Q) Given ar[N], where every element repeats twice except for 1 element, which occurs once, find that unique element.

Constraints:

$1 <= N <= 10^5$

$1 <= ar[i] <= 10^9$

Ex1: ar[5] = 6 ^ 9 ^ 6 ^ 10 ^ 9 = 10 (uni)

0	1	2	3	4	5	6	uni
2	^ 3	^ 5	^ 6	^ 3	^ 6	^ 2	= 5

Ex2: ar[7] =

Idea: Calulate XOR of all elements

```
int unique(int ar[]){ TC: O(N)  SC: O(1)
    int N = ar.length;
    int val = 0; //0 ^ x = x : 0 won't effect your ans
    for(int i=0; i<N; i++){
        val = val ^ ar[i]
    }
    return val;
}
```

0	1	2	3	4	
6	^ 9	^ 6	^ 10	^ 9	= 10 (ans)

Ex1: ar[5] =

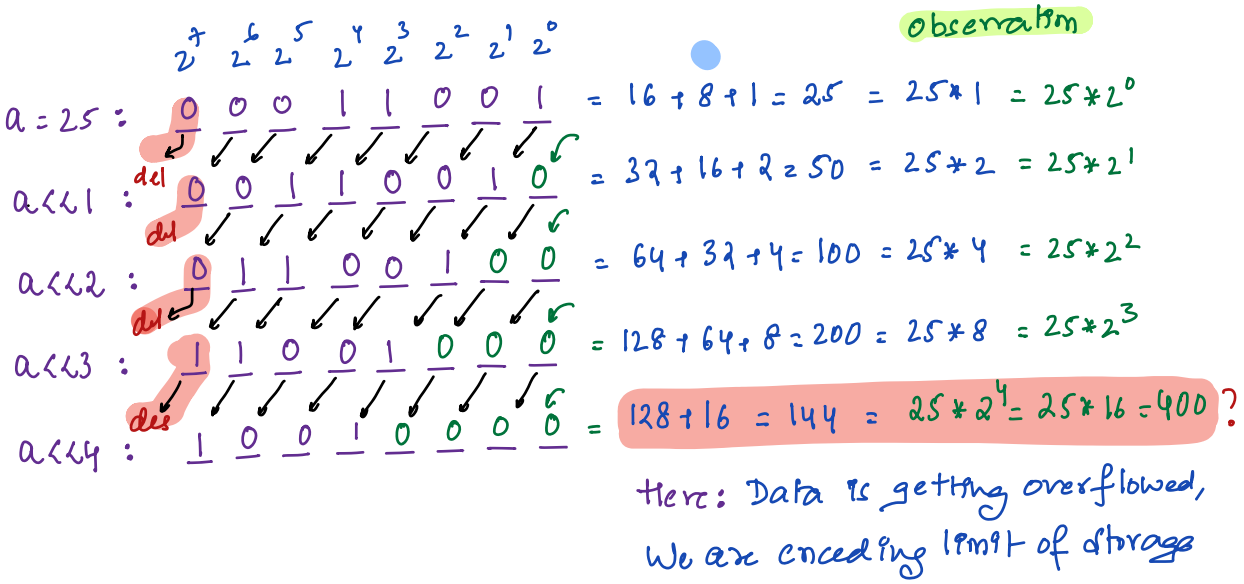
val	i th : val = val ^ ar[i]	val
0	0 : val = val ^ 6	0^6 = 6
6	1 : val = val ^ 9	6^9 = 15
15	2 : val = val ^ 6	15^6 = 9
9	3 : val = val ^ 10	9^10 = 3
3	4 : val = val ^ 9	3^9 = 10

10 → unique element

	2^3	2^2	2^1	2^0	
6:	0	1	1	0	
9:	1	0	0	1	
15:	1	1	1	1	
6:	0	1	1	0	
9:	1	0	0	1	= 9
10:	1	0	1	0	
3:	0	0	1	1	= 3
9:	1	0	0	1	
	1	0	1	0	= 10

leftshift <<

Say a is 8 bit number? At max it can store 8 bits



Here: Data is getting overflowed, We are exceeding limit of storage

obs: a<<n = a * 2^n : if no overflow

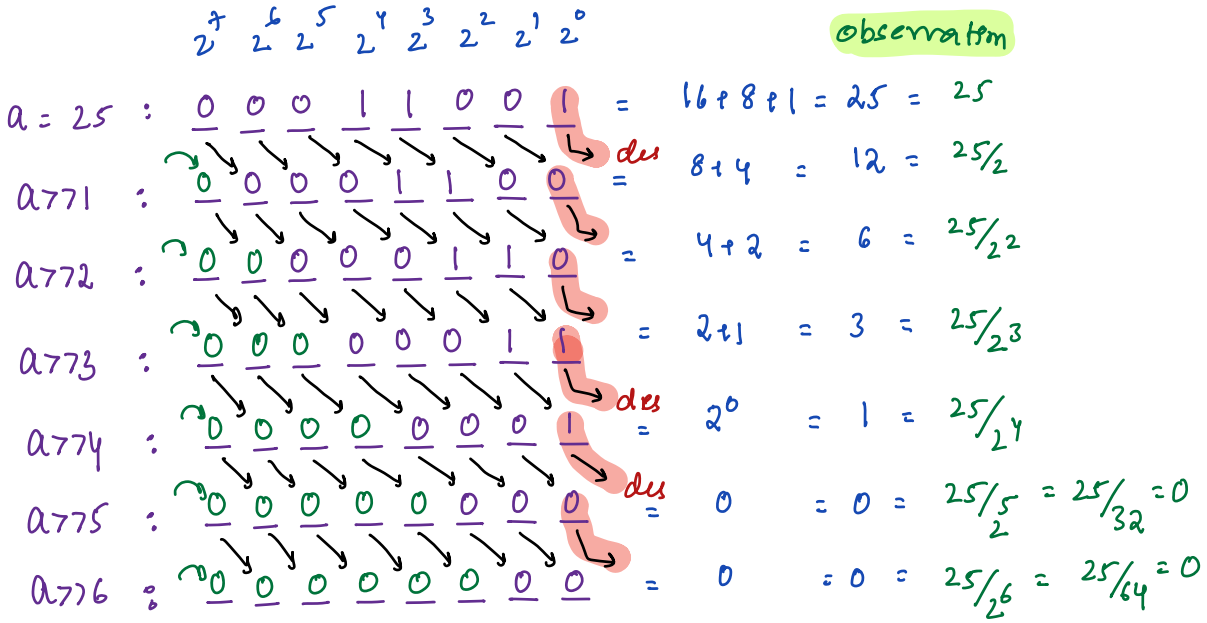
1<<n = 1 * 2^n = 2^n

5<<3 = 5 * 2^3 = 5 * 8 = 40

When overflow occurs : Learn in Bit Manipulations 2

Rightshift >>

Say a is 8 bit number.



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

$40>>3 = 40/2^3 = 40/8 = 5$