BiT Manupulations

lecture 1

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

2^{2} seen

2^{3} seen

operators

&

a = 5 b = 6

```
  101
& 110
-----
  100   = 4
```

a = 5 b = 7 c = 8

a & b & c

```
  101
  111
 1000
 ----
 0000
```

```
a -> 101  5
b -> 111  7
-----
  111
```

^ (XOR)

```
  5  7
  101
  111
  ---
  010
```

~ (Negation)

1 -> 0

0 -> 1

Right shift (>>)

a = 5

10.1

a >> 1

$10_2 \Rightarrow 2$

(5 >> 1) -> 2

$\frac{5}{2} = 2$

a = 10

1010

a >> 3

$1_2 \Rightarrow 1$

(10 >> 3) -> 1

$\frac{10}{2} = \frac{5}{2} = \frac{2}{2} = 1$

1010

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

0001

1×2^0 / 2^3

$$(>> num) = \frac{\text{decimal no}}{2^{num}}$$

left shift (<<)

$(5)_{10} \rightarrow (000\ldots0101)_2$

32 bits.

int a = 5

↳ 4 bytes

32 bits

(5 << 2)

$(10100)_2$

1×2^2

101

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

$\times 2^2$

(5×2^2)

Q:- arr[] = { 2, 1, 2, 5, 6, 5, 7, 7, 6 }

Array contains N integers -> Find the number which occurs only

Sol:

```
XOR = 0
for (i=0; i<n; i++)
{  XOR = XOR ^ a[i];
}
Print (XOR).
```

Q: Swap 2 numbers (imp for interview - without using Third variable).

a = 5
b = 7

① a = a^b
② b = a^b
③ a = a^b

①	②	③
a = 5^7	b = 5^7^7	a = 5^7^5
b = 7	a = 5^7	b = 5

Q: Given N, print the XOR of all numbers b/w 1 to N

ex: 5 1^2^3^4^5

ans:

```
XOR = 0
for (i=1 -> N)
    XOR = XOR ^ i;
Print (XOR)
```

O(N)

N = 1	1
N = 2	3
N = 3	0
N = 4	4
N = 5	5
N = 6	7
N = 7	0
N = 8	8

if (n%4==0) Print(n)
if (n%4==1) Print(1)
(n%4==2) (n+1)
(n%4==3) 0

O(1).

Q: Given a range (L-R), Print the XOR (L^L+1^ ..... R-1^R)

Sol: L = 2, R = 4 ; 2^3^4

another way Time Complexity

(3 - 6)

XOR(6) XOR(2) -> 3^4^5^6

(1^2^3^4^5^6) ^ (1^2)

3^4^5^6

XOR (R) ^ XOR (L-1) -> O(1)

Q: even (or) odd

if (n % 2 == 0) even else odd — more Time

if (n & 1 == 0) even else odd — less Time

12 -> 1100 -> 1100 & 1 = 0000

13 -> 1101 & 1 = 0001

Q: check if The ith bit is set (or) not.

N = 13 -> $(1101)_2$ (bits 2, 1, 0 marked)

i = 2 yes

i = 1 no

	4	3	2	1	0	
	1	1	0	0	1	
mask ->	0	1	0	0	0	(mask & n)
&	0	1	0	0	0	!= 0 yes (1<<3)

mask = (1<<i)

	1	0	0	0	1
&	0	1	0	0	0
	0	0	0	0	0 -> 0 no

bool set = (mask & n)

Q: extract i-th bit of The number

n = 13 i = 3 (Same as before)

3 2 1 0
1 1 0 1

if yes 1

no 0

Q: Set The ith bit of a number.

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

i = 2

Soln

	5	4	3	2	1	0
	1	1	0	0	1	0
mask	0	0	0	1	0	0
'or' / operation	1	1	0	1	1	0

mask = (1<<i)

n = n | mask

Q) Clear The ith bit

```
5 4 3 2 1 0
1 1 0 0 1 0        i = 2

110010     already '0'
           0 -> 0

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

100010     if 1 -> 0
```

ans:

```
    ↓
  1 1 0 0 1 0
& 1 1 1 0 1 1      ~(000100)     mask ~(1<<i)
  -----------
  1 1 0 0 1 0                    (n & mask)
  -----------
```

Q) Remove The last set bit

```
      last
  1 1 0 1 1 0
  1 1 0 1 0 0
```

(n&n-1)

```
       last
12 -> 1100
11 -> 1011
      ----
      1000
      ----

   last
  110110
  110100
  110000
```

Q) Check if number is power of 2

if (n&n-1==0) ✓

else X

```
n   1000
n-1 &0111
    -----
    0000
```

Q) count the no.of set bits.

```
      3210
14 -> 1110   -> set bits (3)

4321
1110
4 -> no. of bits.
O(MSB)
Position Most significant bit
```

```
cnt = 0.
while (n! = 0)
{
   if(n&1 == 1)
     cnt++;
   n = n>>1;
}
print(cnt);
```

```
14-> 1110      0111
     &1        &1
     ----      ----
     0000      0001
               cnt++
               = 1

1110
->0111
```

another way   n=13   1101 ①
1100 ②
1000 ③
0000

```
cnt=0;
while (n!=0)
{
   n = n& (n-1);
   cnt++;
}
print(cnt);
```

O(set bits)
slighly optimal for some.

n=15   1111
both methods
same time complexity.

lecture 2

Q:- N integers -> every integer appears twice two integers appear once.

[1, 1, 2, 5, 3, 2, 3, 4, 7, 4]

5, 7 answer

Sol:- Brute

```
for (i=0; i<n; i++)
{ cnt=0
  for (j=0; j<n; j++)
  {
     if(a[j]==a[i]) cnt++;
  }
  if (cnt==1) print (a[i]);
}
```

O(N^2)

Map

```
map<int, int> mpp;
for (i=0; i<n; i++)
{
   mpp[a[i]]++;
}
for (auto it: mpp)
{ if (it.second==1)
     print(it.first).
}
```

TC-> O(N(logn)
SC-> O(N).

n + map logn
unordermap O(1)
O(N) worst.

another arr[] -> [2, 1, 2, 5, 1, 4, 4, 7, 3, 3]

XOR = 5^7 = 2
(2 1 0)
101
^ 111
0(1)0

for (i=0->n)
XOR = XOR ^ a[i]

O(N)

5 (1st index) was 0 | 7 (1st index) was 1

~~1~~ | ~~2~~
~~5~~ | ~~2~~
~~1~~ | 7
~~4~~ | ~~3~~
~~4~~ | ~~3~~
/5/ | /7/

XOR = 0
for (i=0->n)   XOR=2.
    XOR = XOR^a[i];

    cnt = 0
    while(XOR)
O(32) {
        if (XOR&1) != 0
            break;
        else
        cnt++;   (cnt=1)
        XOR = XOR>>1;   index=1
    }

0 1 0
& 1
000

0 0 1
& 1
0 0 1 -> (1) != 0

XOR1=0   XOR2 = 0.

for (i=0->n)
O(N) { if ((a[i] & (1<<cnt)) != 0)
            XOR1 = XOR1^a[i];
TC=(O(N))
SC-(O(1))   else
            XOR2 = XOR2^a[i];
    }
    print(XOR1)
    print(XOR2).

Q) Generate all the subsets   arr-> [3, 2, 4].   (index 0 1 2)
Power set   n=3
                                   2^3 -> 8

(bit index)

num	2	1	0	
0	0	0	0	-> { }
1	0	0	1	-> {3}
2	0	1	0	-> {2}
3	0	1	1	-> {3, 2}
4	1	0	0	-> {4}
5	1	0	1	-> {3, 4}
6	1	1	0	-> {2, 4}
7✓	1	1	1	-> {3, 2, 4}

8X -> (2^n)-1 -> (1<<n) - 1
n=3   (1000) - 1
             (0111 -> 7)

for (num=0 -> (1<<n)-1)
{
    ds = [ ];   empty
    for (bit=0->n-1)
    { if(num&(1<<bit))
            ds.add [a[bit]]
    }
    for(auto it: ds)
    { print (it);
    }
};

(TC = $2^n \times n$)

This algo only runs for n ≈ 16/17/18
> This may be TLE