Session - 3 (modulo depends on compiler, language)

$$(a+b) \% m = (a \% m + b \% m) \% m$$

$$(a-b) \% m = (a \% m - b \% m) \% m \quad \times$$

$(20-9) \% 6 = (20 \% 6 - 9 \% 6) \% 6$

$5 \neq -1$ ☒

$$(a-b) \% m = (a \% m - b \% m + m) \% m$$

$(20-9) \% 6 = (20 \% 6 - 9 \% 6 + 6) \% 6$

$5 = 5$ ☑

$$(a * b) \% M = (a \% m * b \% m) \% m$$

$$(a / b) \% m = \left(\frac{a \% m}{b \% m}\right) \% m$$

Denominator may become zero.

Modulo distribution over Division (TODO)

→ Extended Euclid's Algorithm

```
int a = 1e6, b = 1e7

int c = a * b;   // overflow
```

1e13 ← ✗

so, long long = a * b (no! It'll not work)

int * int = int not long long

↓

(Typecasting is the sol'n)

```
long long a = 1e6, b = 1e7;
long long c = a * b;
```
✓ (It'll work)

```
int a = 1e6, b = 1e7;
long long c = (long long) a * b;
```
✓ (works)

If It is overflowing, check Data types,

Do Type casting

Else,

go with modulo (if mentioned in Question)

For any problem, while finding solutions we go with "Time complexity" and "space complexity".

---

(1)

N	i/p	o/p Divisors of N (count)
12		1, 2, 3, 4, 6, 12 (6)
5		1, 5 (2)
24		1, 2, 3, 4, 6, 8, 12, 24. (8)

```
int divisors (int N) {
    int c = 0;
    for (int i = 1; i <= N; i++)
        if (N % i == 0)
            c++;
    return c;
}
```
} N

Constraints : $1 \leq N \leq 10^9$

N iterations $\cong$ 10^9 iterations

1 GHz = 10^9 instructions / sec

10^9 instructions → 1 second

let us,

1 iteration $\cong$ 5 instructions

⇒ 5 seconds

$1 \leq N \leq 10^{18}$

10^9 iterations → 1 sec

10^{18} iterations → 10^9 seconds

$\frac{10^9}{60 \times 60 \times 24 \times 365}$ yrs $\cong$ 31.7 yrs (approx)

31.7 × 5 yrs > 150 yrs.

(Importance of optimization)

24

(i)	24 (n/i)
1	24
2	12
3	8
4	6
6	4
8	3
12	2
24	1

(Repeating)

$i \leq \frac{N}{i}$

100

(i)	(N/i)
1	100
2	50
4	25
5	20
10	10
20	5
25	4
50	2
100	1

(repeating)

$i \leq \frac{n}{i}$

$i * i <= N$

$\boxed{i \leq \sqrt{N}}$

N → previously 10^{18} iteration → 10^9 sec

$\sqrt{N} \rightarrow \sqrt{10^{18}} = 10^9$ iterations → 1 sec

$\simeq 1 \times 5$ sec

Optimized :

```
int divisors ( int N )
{
    int cnt = 0;
    for ( int i = 1 ; i*i <= N ; i++ )
    {
        if ( N % i == 0 )
        {
            if ( i == N/i )
                cnt++;
            else
                cnt += 2;
        }
    }
    return cnt;
}
```

```
void Solve ( int N ) {
    int a = 20, b = 40;
    long long c = 10.0;
    float z = 2.5;
```

```
for ( int i=0; i<5; i++) { 5
{       a = b+c;
}
}
```

If we change N, nothing will change

constant time comp. O(1).

(constant time)

→ Eliminate lower order terms.

```
for ( i = 0 to N)
{     for (i = 1 to 5)
      {
      }
}
```

N * 5

O(N)

(Linear Time)

→ $O(N^2)$

```
for ( i = 0 to N)
{     for (j=0 to N)
}
```

N^2.

```
for ( i=0 to N)
{
}
```

N

$= O(n^2) + O(n)$

$= O(n^2+n)$

Eliminate

$(N^2 + N)$

=

0% of N, $N = 10^3$

$\frac{N}{N^2} \times 100$

$\frac{10^3}{10^6} \times 100$

$= 0.1$

$N = 10^4$,

$\frac{10^4}{10^8} \times 100$

$= 0.01$

$N^2 + \frac{N}{2}$

↳ Contribution is very less, so remove

$\Rightarrow O(N^2)$

```
for ( i=0; i<N; i=i*2)
{
    .....
}
```

$0 * 2 = 0$ ∞∞

Infinite loop (TLE)

```
for ( i=1; i<=N; i=i*2)
{
  ...
}
```

$\log_2 n$

$=$

	i	total
2^0	1	1
2^1	2	1 + 2
2^2	4	1 + 3
2^3	8	1 + 4
	⋮	
	2^K	$= (K+1)$

$=$

$2^K = N$

$\log_2 2^K = \log_2 N \Rightarrow \boxed{K = \log_2 N}$

Sol 1 : $4n^2 + 3n + 10040$

Sol 2 : $n^3 + n$

which is Better

	$n = 10$	$n = 10^3$	$n = 10^4$	$n = 1$
Sol 1	10430	$4 \times 10^6 + \cdots$ ✓	$4 \times 10^8 + \cdots$ ✓	100047
Sol 2	1010 ✓	10^9	10^{12}	2. ✓

upto certain threshold 2 is performing better, after that 1 is better

---

Sol 1 : $10n^2 + n \sim O(n^2)$

Sol 2 : $n^2 + 10n \sim O(n^2)$

	1	10^2	10^4
Sol 1	"	$10^5 + \cdots$	$10^9 + \cdots$
Sol 2	"	$10^4 + \cdots$ ✓	✓ $10^8 + \cdots$

(2) is better

Linear search

```
bool lin_search ( int ar[], int N, int Req)
{
O(N)  for ( int i=0; i<N; i++)
      {   if ( ar[i] == Req )
              return 1;
      }
      return 0;
}
```

Best : O(1)

Aug : O(N/2)

worst : O(N)

↑ — upper Bound

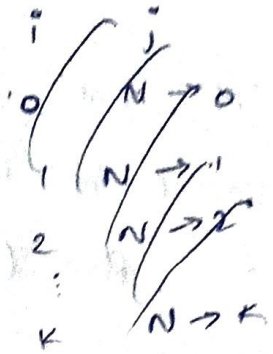
Big-oh Definition

upper Bound
threshold
input size

Puts an upper Bound on complexity of an algorithm based on input size after a certain threshold.

⇒

```
int a = 0;
for ( i = 0 to N )
    for ( j = N; j > i; --j )
    {
        ...
    }
}
```



i	j	total
0	[N, 0)	N − 0 = N
1	[N, 1)	N − 1
2	[N, 2)	N − 2
3	[N, 3)	N − 3
⋮	⋮	⋮
N − 1	[N, N − 1]	1

$1 + 2 + 3 + \cdots N = \frac{N(N+1)}{2} = \frac{N^2}{2} + \frac{N}{2}$

$\Rightarrow O(N^2)$

[a, b] = b − a + 1

[2, 5] = 5 − 2 + 1 = 4

[a, b) = b − a

(a, b] = b − a

(a, b) = b − a − 1

⇒

```
int a = 0, i = N;
while ( i > 0 )
{
    a += i;
    i /= 2;
}
```

i	total
$N/2^0$ N	1 = 1
$N/2^1$ N/2	+ 1 = 2
$N/2^2$ N/4	+ 1 = 3
⋮	⋮
$N/2^K$	K + 1

$\log_2 N$

$$\frac{N}{2^k} = 1 \Rightarrow N = 2^k$$

$$k = \log_2 N.$$

```
int cnt = 0;
for (int i = N; i > 0; i /= 2)
{   for (j = 0; j < i; j++)
        cnt += 1;
}
```

i	j	total
$N/2^0$	N	$[0, N)$	N
$N/2^1$	$N/2$	$[0, N/2)$	$N/2$
	$N/4$	$[0, N/4)$	$N/4$
	⋮		
$\frac{N}{2^k}$		$[0, \frac{N}{2^k}]$	$N/2^k$

$$\frac{N}{2^k} = 1 \Rightarrow k = \log_2 N.$$

$$N + \frac{N}{2} + \cdots \frac{N}{2^k}$$

$$= N\left[1 + \frac{1}{2} + \frac{1}{2^2} + \cdots \frac{1}{2^k}\right]$$

$$N \cdot 1\left[1 - \left(\frac{1}{2}\right)^{k+1}\right] \Big/ \left(1 - \frac{1}{2}\right)$$

$$= 2 \cdot N \cdot \left[1 - \frac{1}{2}\right]^k$$

$$= 2N$$

$$= O(N)$$

$$a\left[1 - r^n\right] \Big/ \left[1 - r\right]$$

```
int k = 0;
for ( int i = N/2 ; i <= N; ++i)
    for ( j = 2 ; j <= N; j = j * 2)
        {  ...
        }
```

$N/2$ ← outer loop

$\log_2 N$ ← inner loop

$O(N \log_2 N)$

```
for ( i = 1; i <= N; i++)
{   int p = pow (i, k);
    for ( j = 1 ; j <= p; ++j
        {  ...
        }
}
```

i	j	total
1	$[1, 1^k]$	1^k
2	$[1, 2^k]$	2^k
3	$[1, 3^k]$	3^k
⋮	⋮	⋮
N	$[1, N^k]$	n^k

$= 1^k + 2^k + \cdots + N^k$

$k = 1, \quad 1 + 2 + \cdots N = \frac{N(N+1)}{2}$

$= \left[\frac{N^2}{2}\right] + \frac{N}{2}$

$k = 2, \quad 1^2 + 2^2 + \cdots + N^2 = \frac{N(N+1)(2N+1)}{6}$

$= \frac{2N^3}{6} + \cdots$

$= \boxed{\frac{N^3}{3}} + \cdots$

$k = 3, \quad = \boxed{\frac{N^4}{4}} + \cdots$

$\vdots$

$k \rightarrow \frac{N^{k+1}}{(k+1)}$

```
j = 0
for (i = 0; i < N; ++i)
{
    while (j < N && arr[i] <= arr[j])
    {
        j++;
    }
}
```

i	j	total
=	=	<=
0	0	

Wrong Answer → logic
Edge cases
Overflow (data type)

TLE → Infinite loop
Brute force Approach

RTE → out of Bound
unhandled exceptions

CE → Syntax Error (read Error)

1 GHz = 10^9 instructions/sec.

1 iteration ≈ 5 instr

$T \times n \leq 10^8$ iterations.

↓
Test case

$T = 10^3$, $N = 10^2$

$T \times N^2 = 10^3 \times 10^4 = 10^7$ ✓

$T \times N^3 = 10^3 \times 10^8 =$ → ✗
TLE

$T = 100$, $N = 20 \rightarrow T \times 2^N$ ✓

$T = 100$, $N = 30 \rightarrow T \times 2^N$ ✗

$T = 100$, $N < 10^8$ →

Importance of constraints

1. Data types
2. Decide the logic

Session - 4

Let us suppose,

Space

```
void Solve (int N)
{
    int a = 10, b = 20;      → 4B + 4B = 8B
    float c = 6.1;           → 4B
    char ar[1000];           → 1B × 1000 = 1000B
    char br[N]               → N B
}
```

Constant space complexity O(1)

O(N) space (Depends on N)

1012B + NB

(neglect constant)

int ar[N]; ⇒ N ⇒ O(N)

int cr[N][N]; ⇒ N × N ⇒ O(N²)

int mat1[N][N], mat2[N][N] → N² + N²

→ 2N²

≅ O(N²)

⇒ Any online platform → Space limit [256MB (or) 128MB]

Suppose,

int ar[N]. $1 \le N \le 10^4$

$1MB = 10^6$

$1KB = 10^3$

ar[10^4] → 4×10^4 B

→ 40KB ✓ (works)

ar[10^6] → 4×10^6 B

4 mB (works)

ar[10^7] → 40 mB (works)

ar[10^8] → 400 mB ⇒ out of space

(400 mB >> 256 mB)

(doesn't work)

Suppose,

bool ar[10^8] → 1×10^8 B = 100 mB (works)

Conclusions : ***

$N \leq 10^{6-7}$ (int) ✓
$N \leq 10^8$ (bool)

=

ar[10^3][10^3] ⇒ 10^6

= 4 mB ✓

ar[10^4][10^4] → 10^8 → 400 mB ✗

bool ar[10^4][10^4] → 10^8 → 100 mB ✓

Conclusions : ***

2D array → $N \leq 10^5$ (int)
$N \leq 10^4$ (bool)

***

Time limit → $N \leq 10^8$

Space limit → $N \leq 10^{6-7}$ (int)

$\leq 10^8$ (bool)

① I/P :

	0	1	2	3	4	5	6	7	8
arr :	5	1	2	12	5	16	1	12	2
N									

Everything is repeated twice
Except one number.

O/P : 16

Sol:1 : Traverse each Element, check whether every Element is repeated or not. [count method]

```
int repeated ( int arr[], int N )
{
    for ( int i = 0; i<N ; i++ )
        int cnt = 0;
        for ( int j = 0 ; j<N ; j++)
        {
            if ( arr[i] == arr[j])
                cnt++;
        }
        if ( count == 1 ) return arr[i];
    }
}
```

TC : $O(N^2)$

SC : $O(1)$

Sol : 2 :

$a \wedge a = 0$

$\boxed{a \wedge b \wedge a = b}$

0, 1, 2, 12, 5, (16), 1, 12, 2

```
int xor = 0;
for (i = 0 → N-1)
    xor = xor ^ arr[i];
return xor;
```

Tc : O(N)

Sc : O(1)

Solutions

1) N^2, 1 [Count method]

2) N, 1 [XOR]

---

(2)

I/P a	I/P b	O/P a^b
5,	2	25
2,	10	1024
3,	4	81

Solutions

1) O(b), O(1)

2) O(31), O(1)

3) $O(\log_2^b)$, O(1)

General way ★ b, 1

loop 'b' times.

$3^{21} \rightarrow 3^{x1} \cdot 3^{x2} \cdot 3^{x3} \ldots$

x_1, x_2, x_3 — Powers of 2, unique

$3^{21} = 3^{16} \cdot 3^4 \cdot 3^1$

$3^{48} \rightarrow 3^{32} \cdot 3^{16}$

$3^{21} = 3^{16} \cdot 3^4 \cdot 3^1$

$21 = 10101$

Bit position

$x = 3$

$x = x * x$

ans = 1

Bit position	$x = x * x$	ans = 1
0	3^1	3
1	3^2	3
2	3^4	3^5
3	3^8	3^{21}
4	3^{16}	

3^{21}

21 = 10101

16 8 4 2 1

Do for $3^{48} = 3^{32} \cdot 3^{16}$ => 110000

Bit pos	$x = x * x$	ans = 1
0	3^1	3
1	3^2	3
2	3^4	3
3	3^8	3
4	3^{16}	3^{16}
5	3^{32}	3^{48}

```
int compute (int a, int b)
{
    int x = a, ans = 1;
    for (i = 0 → 30)
    {
        if (checkBit (b, i))
        {
            ans = ans * x;
        }
        x = x * x;
    }
    return ans;
}
```

(refer less-2)

$\Rightarrow (b >> i \& 1) != 0$

```
int x = a, ans = 1;
while (b > 0) {
    if ((b & 1) != 0)
    {
        ans = ans * x;
    }
    x = x * x;
    b = b >> 1;      (⇒ b = b/2!)
}
return ans;
```

$\log_2 b$

(3) I/P:

ar_N : 5 12 5 12 16 5 14 14

Everything is repeated thrice, Except one number

O/P : 16

Solutions:

1) $O(N^2)$, $O(1)$

2) $O(31*N)$, $O(1)$

Sol 1 : [previous count method]

```
for ( i = 0 to N)
{       cnt = 0
        for ( j = 0 to N)
            if (ar[i] == ar[j]
                cnt++;
        if ( cnt == 1). return ar[i];
```

3

Sol 2:

5 — 00101

12 — 01100

5 — 00101

12 — 01100

16 — 10000

14 — 01110

5 — 00101

14 — 01110

12 — 01100

14 — 01110

cnt

0th pos set 5,5,5 (3)

unset 12,12,12,14,14,16 (7)

0

1st pos set 14,14,14 (3)

unset 5,5,5,12,12,12,16 (7)

0 0

2nd pos set 5, 5, 5, 12, 12, 12, 14, 14, 14 (9)

unset 16 (1)

0 0 0

3rd pos set 12, 12, 12, 14, 14, 14 (6)

unset 5, 5, 5, 16 (4)

0 0 0 0

4th pos set 16 (1)

unset 5, 5, 5, 12, 12, 12, 14, 14, 14 (9)

0 0 0 0 = 16

We are checking Every n^{th} position in all numbers, dividing which numbers are set & unset at that particular position, then Based on ~~to~~ (in set / unset).

Set Count:

```
int tripleCount(int arr[], int N)
{
    int ans = 0;
    for (i = 0; i < 31; i++)
    {
        int set = 0;
        for (int j = 0; j < N; j++)
        {
            if (checkBit(arr[j], i))
                set++;
        }
        if (set % 3 != 0)
        {
            ans = ans | (1 << i);
        }
    }
    return ans;
}
```

(4)

	0	1	2	3	4	5	6	7	8	9
arr N :	5	3	2	5	1	6	4	7	8	4

N.

Range [1, N-2] [1, 8]

Two numbers are repeating twice

o/p : 4, 5

Sol 1 : O(2) = O(1) [space]

```
* int arr[2]; int k = 0;
  for ( int i = 0; i < N; i++)
      for ( int j = i+1; j < N; j++)
          if ( arr[i] == arr[j])
              arr[k] = arr[i];
              k++;
          }
  }
return arr;
```

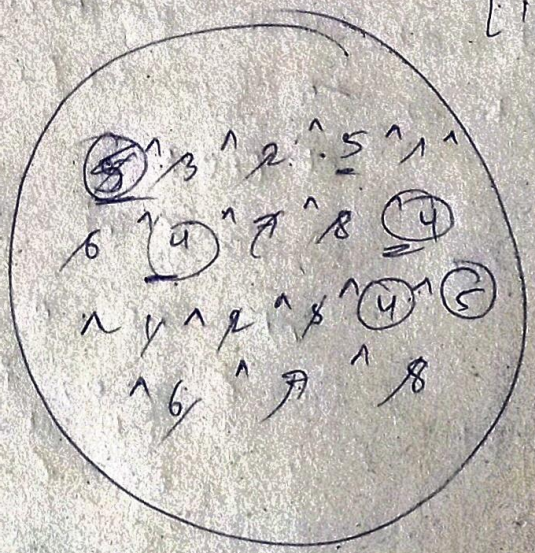
Solutions :

1) O(N²), O(1)

2) O(N), 1
   (O(4N + log₂?))

[1, N-2]
[1, 8]

5 — 0 1 0 1
3 — 0 0 1 1
2 — 0 0 1 0
5 — 0 1 0 1
1 — 0 0 0 1
6 — 0 1 1 0
4 — 0 1 0 0
7 — 0 1 1 1
8 — 1 0 0 0
4 — 0 1 0 0



XOR Bag

int x = 0;

* N-1 times → x = x ^ arr[i]
  i = 0

* N-2 times → x = x ^ i;
  i = 1

$x = 4 \wedge 5$ ✓ How to seperate them ?

4 : 0 1 0 0

^ 5 : 0 1 0 1

4^5 : 0 0 0 1 = 1 (0th bit is seperate)

1 bit which is different in both numbers.

```
int pos = 0;
while (x > 0)
{
    if ((x & 1) != 0)
    {
        return pos;
    }
    pos++;
    x = x >> 1;
}
```

0 || 0

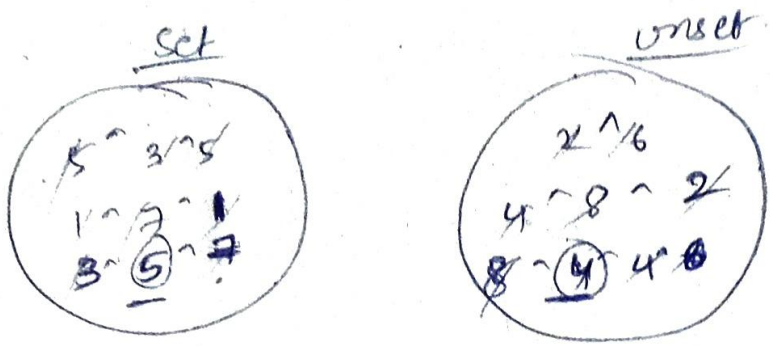
pos = 0

pos = 1

0 | 1

1 2 3 4 5 6 7 8 3[1, N-2]

(4) 0th Bit



6

★ code

step 1: Prev. page

~~step 1~~

z = bit position.

Step 2

```
int a = 0, b = 0;

for ( i = 0 to N-1)
    if ( set)        check bit
        a ^= ar[i]
    else
        b ^= ar[i]

for( i = 0 to N-2)
    if ( set)
        a ^= ar[i]
    else
        b ^= ar[i];

return a, b;
```

$(4N + \log_2 z)$

(5)

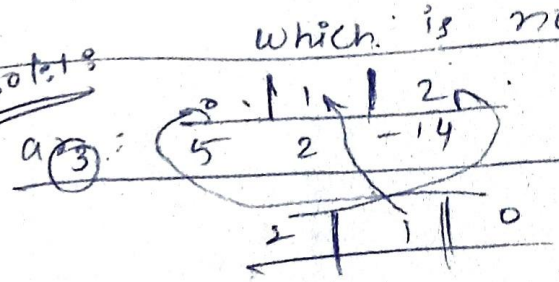
arr N → 5 2 12 -6 20 3 4

K = 10 → True

Subset whose sum = K { 5, 2, 3 }
{ 12, -6, 4 }

Part of an array which is not contiguous.

Solution



1) $O(2^N * N)$, 1

2^3

0 : 0 0 0 → ϕ
1 : 0 0 1 → { 5 } = 5
2 : 0 1 0 → { 2 } = 2
3 : 0 1 1 → { 5, 2 } = 7
4 : 1 0 0 → { -14 } = -14
5 : 1 0 1 → { 5, -14 } = -9
6 : 1 1 0 → { 2, -14 } = -12
7 : 1 1 1 → { 5, 2, -14 } = -7

arr
If N → 2^N subsets (0 to $2^N - 1$) [How to generate subset]

```
for (int i = 0; i < (1 << N); i++)
{
    for (j = 0; j < N; j++)
    {
        if (checkbit(i, j))
        {
            sum += arr[j];
        }
    }
    if (sum == K)
        return true;
}
return false;
```

(however this is not optimized)

```
int ans = 0;
for (i=0; i<N; i++)
{
    ans ^= (2 * arr[i]) // or
                ans ^= (arr[i] + arr[i])
}
```

↓

(N, 1)

---

7)

```
int ans = 0;                          → (N², 1)
for (i=0; i<N; i++)
{
    for (j=0; j<N; j++)
    {
        ans += (arr[i] ^ arr[j])
    }
}
```

5	12	7

x1 x2 x3

5^5 + 5^12 + 5^7 +

12^5 + 12^12 + 12^7 +

7^5 + 7^12 + 7^7

ans = x1 + x2 + x3

5^12 8^7 12^7

22 = 9 + 2 + 11

1001 0010 1011

→ 8+1 2+1 8+2+1

→ 1×2⁰ + 1×2¹ + 0×2²+

1×2³ + 0×2⁴+

0×2⁵+....

3 2 1 0

5 : 0 1 0 1

12 : 1 1 0 0

7 : 0 1 1 1

0th pos — set — 5, 7 ②
unset — 12 ① ⇒ 2 × 1 × 2^0 = 2

1st pos — set — 7 ①
unset — 5, 12 ②
⇒ 1 × 2 × 2^1 = 4

2nd pos — set — 5, 12, 7 ③
unset — — ④ ⇒ 3 × 0 × 2^2 = 0

3rd pos — set — 12 ①
unset — 5, 7 ② ⇒ 1 × 2 × 2^3 = 16

22

22 × 2 = 44

(lower ∆ le also)

```
int ans = 0;
for ( i = 0; i < 31; i++)
{       int set = 0;
        for( int j = 0; j < N; j++)
        {   if ( checkbit ( arr[j], i))
                set++;
        }
        ans += (set) * (N - set) *
                (1 << i);
}
return 2 * ans;
```

$O(31 * N, 1)$