30/9/23

i/p		o/p
a	b	ab
5	2	25
2	10	1024
3	3	27

```
int compute (int a, int b) {
    if (a != 0):
        return a**b
}
```

```
ans = 1
for (int i = 0; i < b; i++)
{
    ans = ans * a;
}
return ans;
}
```

If b ranges 1 to 10^9+7 for large values.

In code $1e9+7$ $\quad ans = (ans * a)\%M$

$1 \times 10^9 + 7$

$9002 \rightarrow 9*e3 + 7$

$0.0004 \longrightarrow 4 \times 10^{-4} = \frac{4}{10^4} = \frac{4}{e4} = 4e-4$

$-1.002 \longrightarrow -\frac{1002}{1000} = -\frac{1000+2}{1000} = -\frac{e3+2}{e3}$

$= -(1 + 2e^{-3})$

$= -1 - 2e^{-3}$

$-49.00075 = -\frac{4900075}{100000} = -(4900075/e+5)$

$= -(4900000+75/e+5)$

$= -\frac{49e5 + 75}{e5}$

$= -49 - 75e^{-5}$

$= -49 - 75e-5$

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

$$(4+5) \% 6 = (4 \% 6 + 5 \% 6) \% M$$

$$3 = (4+5) \% 6$$

$$3 = 3$$

$$(a-b) \% M = (a \% M - b \% M) \% M$$

$$(5-4) \% 6 = (5 \% 6 - 4 \% 6) \% 6$$

$$1 = (5-4) \% 6$$

$$1 = 1$$

for -ve cases not satisfied

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

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

$$(5 * 4) \% 2 = (5 \% 2 * 4 \% 2) \% M$$

$$20 \% 2 = 0$$

$$= 0$$

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

Can't work.

To-do

- Modulo division over division
  - → Extended euclid's algorithm.
    - ↳ inverse algorithm

```
int a = 1e6 , b = 1e7
int c = a * b  // overflow
```

int * int

↓

intermediate result int

↓

then "type casting" takes place

```
long long a = 1e6, b = 1e7;
long long c = a * b
```

(or)

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

i/p	o/p
N	divisors of N
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 (i % N == 0):
            c++
    return c
}
```

31.7 yrs

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

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

$i <= N/i$

$i^2 <= N$

$i <= \sqrt{N}$ → 1 sec

```
int divisors(int N){
    int c = 0.
    for(int i=0; i<=√N; i++){
        if(N%i==0 && (N/i)!=i)
            c = c+2
        if(N%i==0 && (N/i)==i)
            c = c+1
    return c
```

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

$n^2 + n$ → remove lower order terms as the contribution is less.

```
Void solve(int N){
    for(int i=0; i<N; i++){
        for(int j=0; j<N; j++)
    }
    for(int i=0; i<N; i++){ ... }
}
```

→ $O(n^2)$

```
Void solve (int N) {
    for (int i=0; i<N; i=i*2) {
    }
}
```

i=0 → 0*2 = 0 — Infinite loop

```
Void solve (int N) {
for (int i=1; i<=N; i=i*2) {
}
}
```

	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

$2^k = N$

$\log_2 N = k$ } $O(\log n)$

Soll : $4n^2 + 3n + 10000$ → $O(n^2)$

Sol2 : $n^3 + n$ → $O(n^3)$ } which is better? ①

upto certain threshold.

Soll : $10n^2 + n$

Sol2 : $n^2 + 10n$ } which is better? ②

→ $O(n^2)$

```
bool linearSearch (int arr[], int key)
    for (int i=0; i< len(arr); i++)
        { if (arr[i] == key)
                return true;
        }
    return false;
```

Big O definition

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

①
```
int a=0, b=0;
for (int i=0; i<N; i++) {
    a = a + rand();
}
for (int j=0; j<M; ++j) {
    b = b + rand();
}
```
O(n+m)

②
```
int a=0, b=0;
for (int i=0; i<N; i++)
    { for (int j=0; j<N; j++) {
        a = a+j;
    }
}
for (int k=0; k<N; k++) {
    b = b+k;
}
```
O(n²)

c) int a=0

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

i	j	total
0	[N,0)	N
1	[N,1)	N-1

$1+2+\cdots+N-1+N$

$\frac{n(n+1)}{2} = O(n^2)$

$[a,b] = b-a+1$

$[a,b) = b-a$

$[a,b] = b-a$

$(a,b) = b-a-1$

d) int a=0; i=N;

```
While (i>0) {
    a += i;
    i /= 2;
}
```

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

$\frac{N}{2^k} = 1$ } stops here

$N = 2^k$

$\log_2 N = k$

$= O(\log N)$

f)

```
int count=0.
for (int i=N; i>0; i/=2) {
    for (int j=0; j<i; j++) {
        C++;
    }
}
```

i	j	total
N	[0,N)	N
N/2	[0,N/2)	N/2
N/4	[0,N/4)	N/4
⋮		
N/2^k		N/2^k

$O(N)$

$O(\log n)$

$N\left[\frac{1\left(\left(\frac{1}{2}\right)^{k+1}-1\right)}{\frac{1}{2}-1}\right] = 2N\left[1-\left(\frac{1}{2}\right)^k\right] \cong N$

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

g) int k=0;

n { for(int i= N/2 ; i <= N; ++i){

logn { for(int j= 2; j <= N; j = j * 2) {

k = k + N/2;

}

}

O(n logn)

i	j	total
N/2	[2, N]	N-2+1 = N-1
N/2+1	[2, N/2]	N/2+1-2+1 = N/2
N/2+1	[2, N]	

h) int j=0;

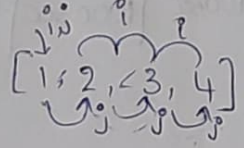
for(int i=0; i<N; ++i)

while(j<N && arr[i] <= arr[j]) {

j++;

}

}



i	j	total
0		

O(N+N)

O(2n) = O(n)

e) void fun(int N, int k){

for(int i=1; i <= N ; i++){

int P = Pow(i, k);

for(int j=1; j <= P; ++j){

- - - - - - }}}

$1^k + 2^k + 3^k + \cdots + N^k$

let k=1

$1 + 2 + 3 + \cdots + N$

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

i	j	total
1	[1, i^k]	$i^k - 1 + 1$ = i^k
2	[1, 2^k]	2^k
⋮	⋮	⋮
N	[1, N^k]	N^k

k=2 $1 + 2^2 + 3^2 + \cdots + N^2 = \frac{n(n+1)(2n+1)}{6} = \frac{2N^3}{6} + \cdots = \frac{2}{3}\frac{N^3}{3} + \cdots$

(k) ; $\boxed{N^{k+1}/k+1}$

1) Linear	N^{k+G}	(3)
2) Logarithmic	$5^{N\times 2}$	(4)
3) Exponential	$\frac{N}{4}\log_2\left(\frac{N}{4000}\right)$	(2)
4) Polynomial	$3^{20}N + 105$	(5)
5) Linear Logarithmic	$10N + 9\frac{N}{100} + 340N^2$	(1)

$10^3 \log_2 (N+3N)$ (2)

1 GHz = 10^9 inst/sec

$N = 10^6$.

Big O.	N^3	$N^2 * \log_2 N$	N^2	$N \log N$	N	$\sqrt{N}\log_2 N$	1
iterations	10^{18}	$10^{12}(19.9)$	10^{12}	$10^6(19.9)$	10^6	$10^3(19.9)$	1
time	31.7yrs	19900sec	1000sec	0.0199 sec	0.001sec	19.9×10^{-6} sec	

N=30	N=60
2^N	2^N
10^9	$(10^9)^2$
1sec	~~31.7yrs~~
	1sec
	31.7yrs

$10^{12}\log_2 10^6$

$10^{12}(19.9)$

$10^9 \times 10^3 (19.9)$

1sec 19900.0

$10^6 \log_2 10^6$

$10^6 (19.9) \times 10^{-6}$

$\sqrt{10^6}\log_2 10^6$

$10^3(19.9)$

Importance of Constraints

1. Data types
2. decide the logic

$2^{10} \approx 10^3$

Space complexity

```
Void Solve (int N) {
    int a=10, b=20;   ⟶ 4B+4B = 8B
    float c=6.1       ⟶ 4B
    char ar[1000];    ⟶ 1B×1000 = 1000B
}                                  1012B
```

O(1)

```
Void solve (int N) {
    int a, b=20
    float c=6.1
    char ar[1000];        } 1012B + NB
    Char br[N];           } O(N)
}
```

$O(N^2)$ ⟶ 2d matrix

even if we declare more than one matrix

i.e $N^2+N^2 = O(2N^2) = O(N^2)$

Space limit (256mb / 128mb)

1Mb = 10^6

1kb = 10^3

arrays $\begin{cases} N \le 10^{6-7} \text{ (int)} \\ N \le 10^8 \text{ (bool)} \end{cases}$

2D array $\begin{cases} N \le 10^3 \text{ (int)} \\ N \le 10^4 \text{ (bool)} \end{cases}$

Time & Space limits

T.L → $N \leq 10^8$

S.L → $N \leq 10^{6-7}$ (int)

$N \leq 10^8$ (bool)

Problem solving

	0	1	2	3	4	5	6	7	8
array of size N:	5	1	2	12	5	16	1	12	2

everything is repeated twice except one number

Solution:

TC	SC
O(n)	O(n)

```
def duplicate(arr):
    for i in arr:
        if i not in arr:
            print(i)
N = int(input())
arr = list(map(int, input().split()))
duplicate(arr)
```

XOR

```
def duplicate(arr):
    for i in range(0, len(arr)):
        c = 0
        for j in range(i, len(arr)):
            if (arr[i] == arr[j]):
                c = c+1
            if (c > 1):
                break
            else:
                print(arr[i])
```

TC	SC
n^2	1

```
def duplicate(arr):
    c = arr[0]
    for i in range(0, len(arr)):
        c = c ^ arr[i]
    return c
arr = list(map(int, input().split()))
duplicate(arr)
```

TC	SC
N	1

$a \wedge a = 0$

$a \wedge b \wedge a = b$

2)

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

eg: $3^{24} = 3^{x_1} \cdot 3^{x_2} \cdots$

$x_1, x_2, x_3 \leftarrow$ Powers of 2 unique.

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

10101

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

bit-posi	x = x * x	ans
0	3	3
1	3^2	3
2	3^4	35
3	3^8	35
4	3^{16}	321
5		

3^{48}

$3^{32}\, 3^{16}$

110000

bit posi	x = x * x	ans
0	3	1
1	32	1
2	34	1
3	38	1
4	3^{16} 3^{32}	3^{16} 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;
}
```

T S
31, 1

bit is set/unset

b: 21 = ---00010101

```
int x=a, bans=1;
while (b>0){
    if ((b&1)!=0){
        ans=ans*x;
    }
    x=x*x;
    b=b>>1;
}
returnans;
```

T S
$\log_2 b$, 1

right shifting

a>>i, a/2^i

③ ar: 5 12 5 12 16 14 5 14 12 14

Everything repeated thrice except one number

o/p: 16.

```
def duplicate(arr):
    for i in range(0, len(arr)):
        c=0
        for j in range(i, len(arr)):
            if (arr[i]==arr[j]):
                c=c+1
                if (c>=3) break
            else:
                print(arr[i])
```

$n^2, 1$

5 12 5 12 16 14 5 14 12 14

00101 01100 00101 01100 10000 01110 00101 01110 01100 01110

0th pos
- set 5,5,5,
- unset 14,14,14,12,12,12,16

set_count % 3 == 0

10000

(16)

1st pos.
- set 14,14,14
- unset 5,5,5,12,12,12,16,

2nd pos
- set 5,5,5,12,12,12,14,14,14
- unset 16

3rd pos
- set 12,12,12,14,14,14
- unset 5,5,5,16

4th pos
- set 16
- unset 12,12,12,14,14,14,5,5,5

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

3(n,1)

(a)

array N: 5 3 2 5 1 6 4 7 8 4

[1, N-2] → write these into array & perform XOR

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

$\sum_{i=0}^{n+1}$ $x = x \wedge ar[i]$ $\sum_{i=1}^{n-2}$ $x = x \wedge ?$ — $n+2$

returns 5, 4

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

n, 1

```
int a=0, b=0
if(set)                        n-2
    a^= ar[i]                  n=1
else b^ ar[i]
```

$\sum_{n=1}^{n-2}$ if(set) $a \wedge = i$; else $b \wedge = i$

retu a, b

(5)

arr = 5 2 12 -6 20 3 4

k = 10

Subset whose sum = 10

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

$(2^n * n), 1$

(6)

```
for(int i=0 i<N, i++)
    for(int j=0; j<N; j++)
        ans^=(arr[i]+arr[j]);
```

$n^2, 1$

[5, -1, 2, 3]

~~5, 7, 8, 3, 4, 5~~

5+5 ∧ 5+1 ∧ 5+2 ∧ 5+3 ∧ 6 ∧ 7 ∧ 8 ∧ 3 ∧ 4 ∧ 5.

1+5 ∧ 1+1 ∧ 1+2 ∧ 1+3.

2+5 ∧ 2+1 ∧ 2+2 ∧ 2+3.

3+5 ∧ 3+1 ∧ 3+2 ∧ 3+3.

5+5 1+1 2+2 3+3

Optimized

```
for i in range (0, len(arr)):
    ans ^= (arr[i] + arr[i])
return ans.
```

O(n)	1

(2)

```
for (int i=0; i<N; i++){
    for (int j=0; j<N; j++){
        ans += (ar[i] ^ ar[j]);
}   }
```

N^2	1

(5, 12, 1)

$5 \wedge 5 + 5 \wedge 12 + 5 \wedge 1 +$

$12 \wedge 5 + 12 \wedge 12 + 12 \wedge 1 +$

$1 \wedge 12 + 1 \wedge 5 + 1 \wedge 1$

$5 \wedge 12 \quad 5 \wedge 7 \quad 12 \wedge 7$

$9 \quad 2 \quad 11$

$8+1 \quad 2 \quad 8+2+1$

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

$2\times2^3 + 0\times2^4 + \cdots$

$---$

```
for (int i=0; i<N; i++){
    if (arr[i] != arr[i+1])
        ans += (arr[i] ^ arr[i+1])
```

5 : 0101

12 : 1100

7 : 0111

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

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

2nd pos — set → 5,7,12 → 0.
unset → 0

3rd pos — set → 12 → 2×1 = $2^1 \times 2^3$ = 16
unset → 5,7

22

22×2 = 44

ans=[], un=0. res=0

```
for (int i=0 ; i<31; i++) {
    for (int j=0; j<N; j++) {
        if (checkbit(ar[j], i)) {
            s++
        }
        if (!checkbit(ar[j], i)) {
            un++
        }
    ans.append(s*un)
return 2*ans.
for i in range(0, len(ans)):
    res = res + 2^i * ans[i]
return 2*res.
```

31*n,1