Starts @ 6:05pm

Q.

```
void fun( int N) {
    int sum = 0 ;
  for ( i = 1 ;  i <= 20;  i++) {

     for ( j = 1;  j <= N;  j++) {

          sum = sum + i * j;
     }
  }

}
```

Table

i	j : [1, N]	iterations
1	[1, N]	N iterations
2	[1, N]	N iterations
⋮		
20	[1, N]	N iterations

20 N iterations

Q.

```
void fun( int N) {
    int sum = 0;
    for ( i = 1; i <= N; i++) {

        for ( j = 1; j <= N; j++) {

            sum = sum + i * j;
        }

    }
}
```

$\Rightarrow N^2$

Table

i	j	iterations
1	[1, N]	N iterations
2	[1, N]	N iterations
⋮		
N	[1, N]	N iteration

N^2 iterations

Q.

```
void fn (int N) {
    sum=0
  for ( i=1; i<=N; i++) {
     for (j=1; j<=i ; j++) {
          sum = sum+ i*j;
     }
  }
}
```

Table

i	j	iterations.
1	[1, 1]	1 iteration.
2	[1, 2]	2 iterations.
3	[1, 3]	3 iterations.
4	[1, 4]	4 iterations.
⋮		
N	[1, N]	N Iterations.

$\Rightarrow 1+2+3+4+5+ \cdots +N \Rightarrow \frac{N(N+1)}{2}$

Bonus Question

Q.

```
void fn (int N) {
    sum = 0
    for (i=1; i<=N; i++) {
        for (j=1; j<=2^i; j++) {
            sum = sum + i*j;
        }
    }
```

Anshu Kumar

Table.

i	j	iterations
1	$[1, 2^1]$	2^1
2	$[1, 2^2]$	2^2
3	$[1, 2^3]$	2^3
⋮		⋮
N	$[1, 2^N]$	2^N

$\Rightarrow \boxed{2^{N+1} - 2}$

$$\text{Total itr} = \overset{a_1}{2^1} + \overset{a_2}{2^2} + \overset{a_3}{2^3} + 2^4 + \cdots + 2^N$$

↳ GP series (Geometric Progression)

↳ Common ratio (r)

$\Rightarrow \frac{a_2}{a_1}, \frac{a_3}{a_2}, \frac{a_{n+1}}{a_n}$

$a_n \to \frac{a_{n+1}}{a_n}$

$\Rightarrow \frac{2^2}{2^1} = 2 \quad \Big| \quad \frac{2^3}{2^2} = 2$

Sum of t terms in gp

$$= \frac{a(r^t - 1)}{r - 1}$$

$\Rightarrow \frac{2^1[2^N - 1]}{2 - 1}$

$\Rightarrow \boxed{2^{N+1} - 2}$

↳ a = first term

r = common ratio

t = no. of terms.

Comparing Algo's using iterations

↳ Problem statement | Input | Testcases

A

↳ $\boxed{100 \log N}$

B

$N/10$

Till $N <= 3500$ A > B

After $N > 3500$ B > A

optimized

B

A

$\boxed{N \log N}$ ←

⇒ **Note:-** We always check algo's performance on large input

Assymptotic Analysis :-

↳ Monitoring performance of algo's for very large input's.

↳ ① $\boxed{\text{Big} (O)}$ ✓

↳ ② Omega (Ω) ✓

↳ ③ Theta (θ) ✓

Big(O)

Steps to calculate Big (O) :

↳ ① Calculate no. of iterations

↳ ② Neglect lower order degree terms

↳ ③ Neglect constant coefficient terms.

Ex:-

$$f(x) = 7x^2 + \frac{5x^4}{x} + 10 \rightarrow O(x^3)$$

$$\Rightarrow \cancel{7x^2} + 5x^3 + \cancel{10}$$

$$\Rightarrow \cancel{5}x^3 \qquad \Rightarrow x^3$$

Q.

① $f(x) = \cancel{7x} + 3x^2 \log x + \cancel{10^4}$

$\Rightarrow \cancel{3} x^2 \log x$

$\Rightarrow O(x^2 \log x)$

② $f(x) = \cancel{2x \log x} + x\sqrt{x} + \cancel{5}$

$\Rightarrow O(x\sqrt{x})$

Note:- ***

$$\boxed{\log(x) < \sqrt{x} < x < x\log x < x\sqrt{x} < x^2}$$

Q. Why neglect lower order terms?

Total itr :- $\boxed{N^2} + 10N$

Input N	Total itr	% contribution of lower order in total
10	$100 + 10*10 = 200$	$\frac{100}{200} * 100 = 50\%$
100	$10^4 + 10^3$	$\frac{10^3}{10^4+10^3} * 100\% = 10\%$
10^4	$10^8 + \boxed{10^5}$	$\frac{10^5}{10^8+10^5} * 100 = 0.1\%$
10^8		

Note:- If input N increases, contribution of lower order term decreases,

Q. Why neglect constant coefficients?

↳ Problem statement | Input | Testcase

$\boxed{\log_2 N < N}$

A	B	
$10 \log_2 N$	N	→ A
$10^2 \log_2 N$	$\frac{N}{10}$	→ A
$10^3 \log_2 N$	$\frac{N}{10^2}$	→ A

Issues with Big O ?

Issue-01

	A	B
	100N	N^2
Big O	O(N)	$O(N^2)$
Input		
N=10	10^3	10^2	B
N = 99	100 * 99	99*99	B
N=100	100 * 100	100*100	Same
N=101	100*101	101*101	A

$$\boxed{\forall\ N > 100}$$

obs: Big O comparison holds only after a certain input value.

Issue - 2

A

$10N^2 + 5N$

Big(O) $O(N^2)$

B

$5N^2 + 10N$

$O(N^2)$ → B

Obs: If 2 algo have same big O, we need iterations to compare.

Space Complexity : ↳ Amount of extra space used by an algorithm

↳ space other than input space.

(Auxiliary space)

array → N , sum

Q. 1

```
void fun () {
    int a = 75;      → 4 byte
    int b = a * a;   → 4 byte
}
```

4 + 4 = 8 byte

O(1) / O(constant)

variable

↳ O(1)

if / else if / break /

return

↳ O(1)

1 int = 4 byte   N int → 4 * N

Q. 2

```
void fun ( int N) {
    int a = 75;      → 4 byte
    int b = a * a;   → 4 byte
    int c [N];       4 * N = 4N
}
```

$4 + 4 + 4N \approx \boxed{4N + 8}$ → itr.

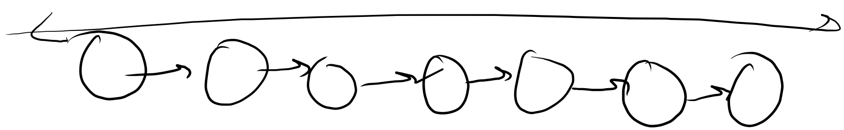
$\boxed{O(N)}$

Q. 3

data → int → 4

address → 4

8 bytes



$\boxed{8N} \approx O(N)$

GCD (Greatest Common Divisor) / HCF (Highest Common Factor)

gcd (a, b) = x : { x is **greatest number** such that a % x == b % x == 0 }

① gcd(15, 25)

15	25
1	1
3	5
5	25
15	

→ 5

② gcd(12, 30)

12	30
1	1
2	2
3	3
4	5
6	6
12	10
	15
	30

→ 6

③ gcd(10, -25)

10	-25
1	-25
2	-5
5	-1
10	1
	5
	25

-25 % -25

(4) gcd(0, 8)

0	8
0	1
1	2
2	4
3	8
4	
⋮	
∞	

→ 8

(5) gcd(0, -10)

0	-10
0	1
1	2
2	5
⋮	10
∞	

→ 10

(6) gcd(-16, -24)

-16	-24
1	1
2	2
4	3
8	4
16	6
	8
	12
	24

→ 8

(7) gcd(-2, -3)

-2	-3
1	1
2	3

→ 1

(8) gcd(0, 0)

0	0
0	0
1	1
2	2
3	3
⋮	⋮
∞	∞

→ Not defined

Properties of gcd(a, b):

→ $\gcd(a, b) = \gcd(b, a)$ ⇒ <u>Commutative</u>

→ $\gcd(a, b) = \gcd(|a|, |b|)$

→ $\gcd(0, x) = |x|$ if $x \mathrel{!}= 0$

→ $\gcd(a, b, c) = \gcd(\gcd(a, b), c)$
$\gcd(\gcd(a, c), b)$
$\gcd(\gcd(b, c), a)$
⇒ <u>Associative</u>

** → if A, B > 0 && A >= B && gcd (A, B) = x

Imp. ↳ $\boxed{\text{gcd}(A-B, B) = x}$

A % x = 0 && B % x = 0

if (A − B) % x = 0 && B % x = 0

↓

(A % x − B % x + x) % x

0 − 0 + 0

$\boxed{\Rightarrow 0}$

5 % 5 = 0

x % x = 0

Claim

** $\boxed{\text{gcd}(A, B) = \text{gcd}(A-B, B)}$

gcd(A − B, B)

↓

gcd(23, 5)

↓

gcd(18, 5)

↓

gcd(13, 5)

→ gcd(23, 5) → gcd(18, 5) → gcd(13, 5) → gcd(8, 5) → gcd(3, 5)

$\boxed{\text{A \% B = A − greatest multiple of B <= A}}$

gcd(23, 5)

↓

1

gcd(3, 5)

↓

1

⇒ $\boxed{\text{gcd}(a, b) = \text{gcd}(a \% b, b)}$ ✓

⇒ gcd(a, b) = gcd(b, a)

⇒ $\boxed{\text{gcd}(a, b) = \text{gcd}(b, a \% b)}$ ✓

→ gcd(a, b) → gcd(b, a % b)