Today's Agenda :-

1. Log Basics + Iteration Problems
2. Comparing Iterations using Graph
3. Time Complexity - Definition and Notations (Asymptotic Analysis - Big O)
4. TLE
5. Importance of Constraints

Basics of Log

$\log_2(64) \rightarrow 6$.

$\log_3(27) \rightarrow 3$

$\log_5(25) \rightarrow 2$

$\log_2(32) \rightarrow 5$.

$\log_b(a) \rightarrow$ what should be power of b to get a.

$\log_b a = c \Rightarrow b^c = a$.

$\log_2(10) \Rightarrow$ 3. something.

$\log_2(40) \Rightarrow$ 5. something.

Note :-

$2^k = N \Rightarrow \log_2 N = k$

$\log_2(2^6) \Rightarrow 6$.

$\log_3(3^5) \Rightarrow 5$

$\log_a(a^N) \rightarrow N$.

Ques.

Given a positive integer N, how many times do we need to divide it by 2 (Consider only integer part) until it reaches 1.

N = 100

$100 \to 50 \to 25 \to 12 \to 6 \to 3 \to 1$ ⇒ 6 times.

N = 324.

$324 \to 162 \to 81 \to 40 \to 20 \to 10$

↓ 8 times.

$10 \to 5 \to 2 \to 1$

$9 \to 4 \to 2 \to 1$ → 3 times.

Generically

$N \to \frac{N}{2} \to \frac{N}{2^2} \to \frac{N}{2^3} \cdots 1.$

$N \to \frac{N}{2^0} \to \frac{N}{2^1} \to \frac{N}{2^2} \to \frac{N}{2^3} \cdots \frac{N}{2^k}$ (k iterations)

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

N = 27, $\log_2 N \to \log_2(27) \Rightarrow 4.$

Quiz 3 :-

How many iterations will be there in this loop ?

```
N>0
i=N;
while(i>1)
{
    i=i/2;
}
```

→ 32

i = 32, 16, 8, 4, 2, 1.

→ $\log_2 N$.

→ $O(\log_2 N)$

Quiz 4 :-

How many iterations will be there in this loop

N = 32

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

($\log_2 N$)

→ $O(\log N)$

1 → 2 → 4 → 8 → 16 → 32

Quiz 5 :-

How many iterations will be there in this loop ?

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

i = 0 → 0 → 0 → 0 → 0 ...

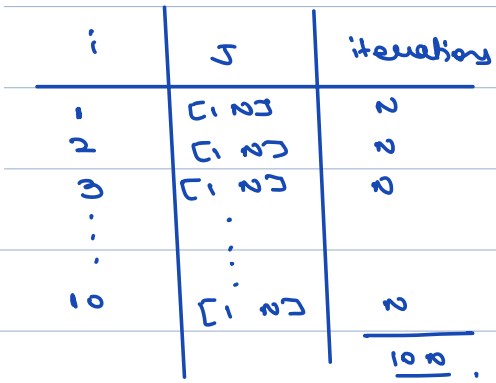
infinite.

Quiz 6 :-

How many iterations will be there in this loop

```
for(i=1; i<=10; i++){
    for(j=1; j<=N; j++){
    /     ......../
    }
}
```

O(n)

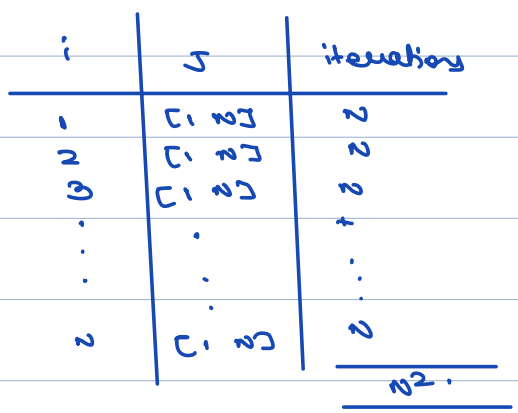


Quiz 7

How many iterations will be there in this loop

```
for(i=1; i<=N; i++){
    for(j=1; j<=N; j++){
        ...
    }
}
```

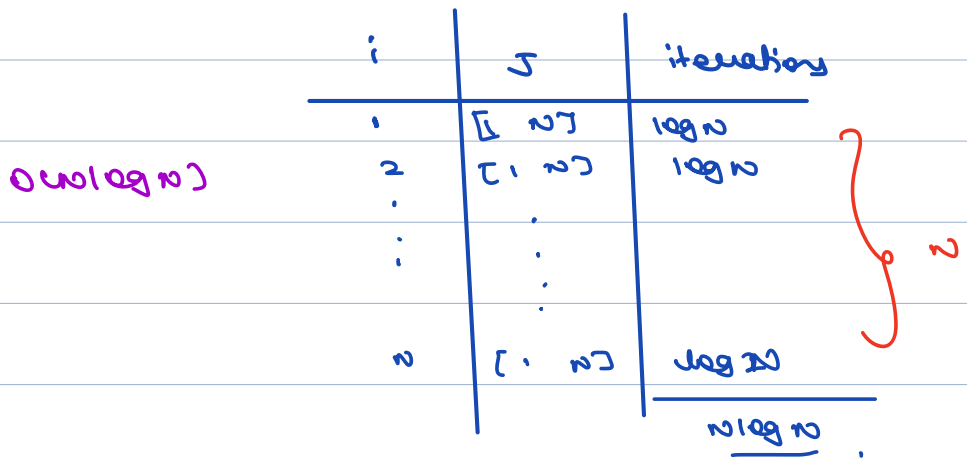
O(n²)



Quiz 8

How many iterations will be there in this loop

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



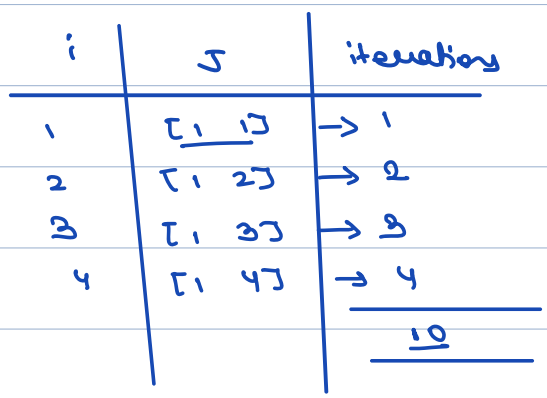
Quiz 9 :-

How many iterations will be there in this loop ?

```
for(i = 1; i <= 4; i++) {
    for(j = 1; j <= i ; j++) {
        //print(i+j)
    }
}
```

iterations are const

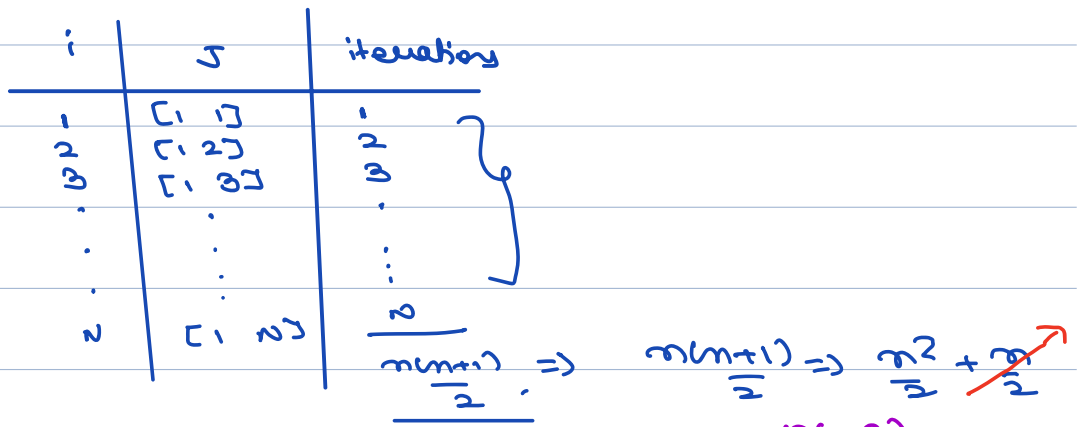
O(1).



Quiz.

How many Iterations will be there in this loop ?

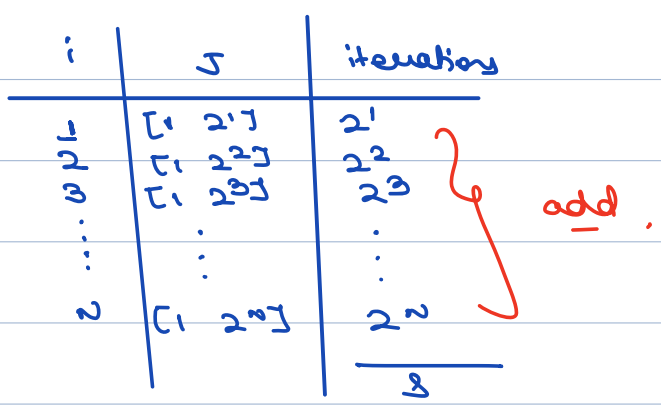
```
for(i = 1; i <= N; i++) {
    for(j = 1; j <= i ; j++) {
        //print(i+j)
    }
}
```



Quiz 11 :-

How many iterations will be there in this loop

```
for(i=1; i<=N; i++){
    for(j=1; j<=(2^i); j++)
    {
        ...
    }
}
```



$S = 2^1 + 2^2 + 2^3 + \cdots 2^N$ $\quad a = 2, r = 2,$ Terms $= N$

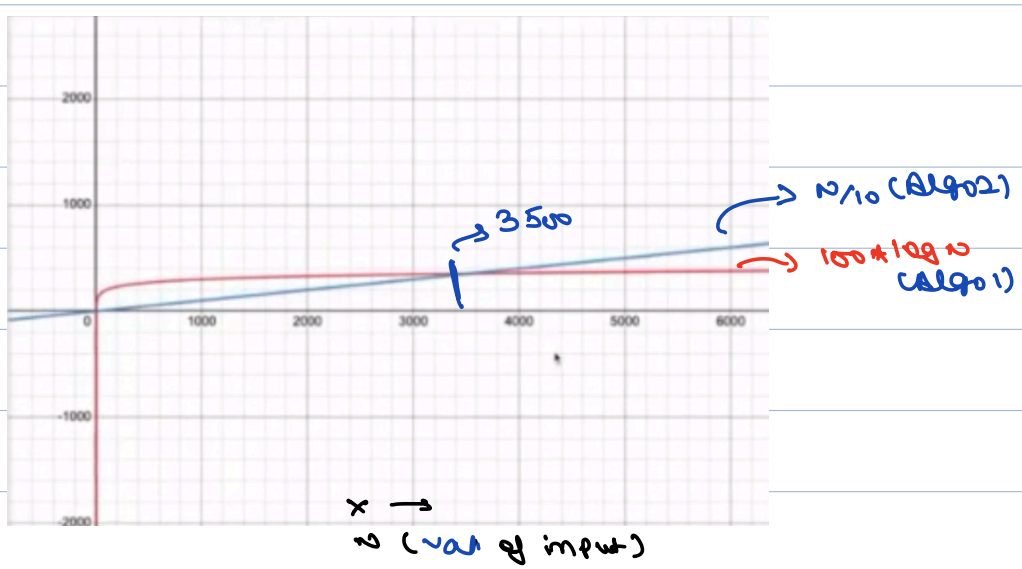
Sum $\frac{a(r^n-1)}{r-1}$ $\Rightarrow$ $\frac{2(2^n-1)}{2-1}$

Total iterations $\Rightarrow 2(2^n-1) \rightarrow 2 \cdot 2^n - 2$.

$\rightarrow O(2^n)$.

-: Compare two different Algorithms :-

	no. of iterations
Algo 1	$100 * \log n$
Algo 2	$n/10$



$n < 3500$ Algo 2 is better

$n > 3500$ Algo 1 is better

In today's world, data is large.

→ 3rd vs Pak → 18 M

→ Baby Shark video → 2.8B views,

↳ we'll say Algo1 is better bcoz its taking less iteration for huge data.

Asymptotic Analysis → Big O

↳ we care in this only when i/p is large.

Big O :

- Calculate **Iterations** based on **Input Size** ✓
- Ignore **Lower Order Terms** ✓
- Ignore **Constant Coefficients** ✓

Algo 1 $100 * \log(n)$ → $O(\log n)$

Algo 2 $n/10$ → $O(n)$

code. → $4n^2 + 3n + 1$ → $O(n^2)$.

$\log(n) < \sqrt{n} < n < n\log n < n\sqrt{n} < n^2 < n^3 < 2^n < n! < n^n$

using an ex, $n = 36$

$5 < 6 < 36 < 36*5 < 36*6 < 36^2 < 36^3 < 2^{36} < 36! < 36^{36}$

Ques) $f(N) = 4N^2 + 3N\log N + 1$

$3N\log N$

$\rightarrow O(N\log N)$.

Ques) $f(N) = 4N\log N + 3N\sqrt{N} + 10^6$

$O(N\sqrt{N})$.

Code 1	Code 2
$O(N^2)$	$O(N)$ ✓

Ques. Why do we neglect lower order terms?

$N^2 + 10N$ iterations

N	Total iterations	Contribution of lower order term
10	200	$100 \Rightarrow$ 50%.
100	$10^4 + 10^3$	$1000 \Rightarrow \frac{1000}{10^4+10^3} \approx 9\%$.
10^4	$10^8 + 10^5$	$10^5 \Rightarrow \frac{10^5}{10^8+10^5} \approx 0.1\%$.

Conclusion :-

we can say as input size increases, the contribution of lower order term decreases.

Ques) why do we neglect constant co-efficients.

Algo 1	Algo 2	for large inputs.
$10 \log_2 N$	N	Algo 1.
$100 \log_2 N$	N	Algo 1
$9 * N$	N^2	Algo 1.
$10 * N$	$N^2/10$	Algo 1.

<- Issues in Big-Oh. ->

Issue-1.

Algo 1 → $10^3 N$ → $O(N)$

Algo 2 → N^2 → $O(N^2)$

	Algo 1	Algo 2	winner
$N = 10$	10000	100	Algo 2
$N = 100$	10^5	10^4	Algo 2
$N = 10^3$	10^6	10^6	Both are same
$N = 10^4$	10^7	10^8	Algo 1 is better

Claim :- for all large inputs $>= 1000$, Algo 1 will perform better else algo 2.

Issue 2 :-

Code 1

```
for(int i=1; i<=N; i++) {
    if(i%2 != 0) {
     c = c+1;
    }
}
```

→ Iterations → N → O(N)

Code 2

```
for(int i=1; i<=N; i=i+2) {
    c = c+1;
}
```

→ Iterations → $\frac{N}{2}$ → O(N).

In both, Big O is O(N), but we know second code is better.

Time Limit Exceeded

Puneet → Amazon → contest

2 questions 1 hour.

1 Question → code → Submit → TLE

↑ changes idea

Can we somehow assess the logic's ability, before we write any code?

Online Editors

↳ 1 GHz machine.

↳ 10^9 instructions per second.

1sec time. Time Limit.

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

Total instructions $\approx 3N$ or $4N$.

Approximation 1 :-

In a small code generally,

1 iteration = 10 instructions

10^8 iterations = 10×10^8 instns

Approximation 2 :-

Suppose you write a big code,

1 iteration = 100 instructions.

10^7 iterations = 100×10^7 instructions

Conclusion :-

Our code can have 10^7 to 10^8 iterations,

then only it'll run in 1sec.

How to approach a problem :-

- Read the **Question** and **Constraints** carefully.
- Formulate an **Idea** or **Logic**.
- Verify the **Correctness** of the Logic.
- Mentally develop a **Pseudocode** or rough **Idea of Loops**.
- Determine the **Time Complexity** based on the Pseudocode.
- Assess if the time complexity is feasible and won't result in **Time Limit Exceeded (TLE)** errors.
- **Re-evaluate** the **Idea/Logic** if the time constraints are not met; otherwise, proceed.
- **Code** the idea if it is deemed feasible.

Constraints

T.C

e.g.1) $1 <= N <= 10^5$.

$O(N^2)$ ✗

$O(N^3)$ ✗

$O(N \log N)$ ✓

$10^5 \log(10^5)$

$(10^5 \times -)$

e.g.2) $1 <= N <= 10^6$.

$O(N^2 \log N)$ ✗

$O(N^2)$ ✗

$O(N \log N)$

$10^6 \times 20$ may or may not be

$O(N)$ ✓

e.g 3) $1 <= N <= 100$, $O(N^3)$ ✓

$(10^2)^3 \approx 10^6$.

e.g 4) $1 <= N <= 20$, $O(2^N)$ ✓

2^{20}

$2^{10} \times 2^{10}$

1024 1024

1000 1000 $\approx 10^6$.