Note: Session starts by 9:05PM Sharp.

Todays Content:

TC1: Calculating iteration in given Codes: 20
Intro to Big O on how to calculate it.

TC2: Time Complenity
Big O
TLE, Time limit Exceed ?

## Basic Maths:

### 1) Sum of first N Natural Numbers:

$$S_N = 1 + 2 + 3 + \dots + N-1 + N = \frac{N*(N+1)}{2}$$

$$S_6 = 1+2+3+4+5+6 = \frac{6*(7)}{2} = 42/2 = 21$$

### 2) Geometric Progression:

Series: 2 → 6 → 18 → 54 → 162 ..

$$\frac{6}{2} = \frac{18}{6} = \frac{54}{18} = \frac{162}{54} = 3$$

Obs: Ratio between 2 adj ele is always same = Geometric Progression.

Series: 3 6 12 24 48 96 ... ratio $r = 2$

Series: 2 10 50 250 ... ratio $r = 5$

Series: 4 12 36 108 ... ratio $r = 3$

$$\frac{12}{4} = \frac{36}{12} = \frac{108}{36} = 3$$

Sum of First T Terms in GP $= \frac{a*[r^T - 1]}{r-1}$

$a$ = 1st Term of GP {geometric progre}

$r$ = common ratio

Series: $2 + 4 + 8 + 16 + 32 = 62$.

Sum of first 5 terms in above series: ——

$a = 2, r = 2, T = 5$: $= \frac{a*[r^T - 1]}{r-1}$

$$= \frac{2*[2^5 - 1]}{2-1}$$

$$= 2*[32-1] = 62.$$

3) Elements in range ?

Elements in range : [3 10] inc corner = 3 4 5 6 7 8 9 10 ans = 8

Elements in range : [a b] (a <= b) inc corner = b - a + 1

Elements in range : [4 7] (a = 4, b = 7) inc corner = b - a + 1 = 7 - 4 + 1 = 4

Note: [ ] indicates corners inclusim :

( ) indicates corners excluded :

Elements in range [a b) # a is inclusive b excluded = b - a

Elements in range (a b) # a is exclud b excluded = b - a - 1

log Basics: 9:25PM

$\log_b^a$ = { To what power we need to raise b to get a }

$\log_2^8 = 3$

$\log_3^9 = 2$

Few Formula

1. $\log_b^{b^N} = N$.
2. if $N = 2^k$, $\log_2^N = k$

Power formula:

1. $a^m * a^n = a^{m+n}$
2. $\frac{a^m}{a^n} = a^{m-n}$
3. $(a^m)^n = a^{m*n}$

Calculate Iterations:

```
void func(int N){
    int s=1;
    for(int i=1; i<=100; i++){
        s = s+i;
    }
}
```

a b b-a+1

i : [1 ... 100] = 100-1+1 = 100.

O(1)

```
void func(int N){
    int s=0
    for(int i=3; i<=50 ; i++){
        print("Hi")
    }
}
```

a b b-a+1

i : [3 ... 50] 50-3+1 = 48

O(1)

```
void func(int N){
    int s=0
    for(int i=1; i<=N; i++){
        print("Hi")
    }
}
```

a b b-a+1

i : [1 ... N] N-1+1 = N = O(N)

```
void func(int N){
    int s=0
    for(int i=0; i<N; i++){
        print("Hi")
    }
}
```

a b b-a+1

i : [0 .. N-1] N-1-0+1 = N. = O(N)

i = N-1 < N

[0 .. N) N-0 = N.

i = N < N

```
void fun(int N, int M)
{
    int s = 0
    for(int i=1; i<=N; i++){
        print("Hi")
    }

    for(int i=1; i<=M; i++){
        print("Hi")
    }
}
```

i : [1 ... N] : N iterations.

Total iterations = N + M = O(N + M)

= O(max(N, M))

i : [1 ... M] : M iterations

```
void fun(int N){
    int s = 0
    for(int i=1; i<=2^N; i++){
        print("Hi")
    }
}
```

a b = b − a + 1

$i : [1 \dots 2^N] = 2^N - 1 + 1 = 2^N = O(2^N)$

```
void fun(int N){
    int s = 0
    for(int i=1; i*i<=N; i++){
        print("Hi")
    }
}
```

$i^* i <= N$

$i^2 <= N$

a b b − a + 1

$i=1,\ i <= \sqrt{N},\ i++)$ $i : [1 \dots \sqrt{N}]$ $\sqrt{N} - 1 + 1 = \sqrt{N} = O(\sqrt{N})$

```
void fun(int N){ // N>0
    int i = N
    while(i > 1)
        print("hi")
        i = i/2
}
```

Examples:

$i = 10 \xrightarrow{i/2} 5 \xrightarrow{i/2} 2 \xrightarrow{i/2} 1$ (i) : Stop

$i = 25 \xrightarrow{i/2} 12 \xrightarrow{i/2} 6 \xrightarrow{i/2} 3 \xrightarrow{i/2} 1$ : Stop

$i = 17 \xrightarrow{i/2} 8 \xrightarrow{i/2} 4 \xrightarrow{i/2} 2 \xrightarrow{i/2} 1$ : Stop

Obs1: When $i = 1$ code stops

| Iterations | Initially $i = N$ |
|---|---|
| After 1 iter | $i = i/2 = N/2 \longrightarrow N/2^1$ |
| After 2 iter | $i = i/2 = \underbrace{[N/2]}_{2} = N/4 \rightarrow N/2^2$ |
| After 3 iter | $i = i/2 = \underbrace{[N/4]}_{2} = N/8 \rightarrow N/2^3$ |
| After 4 iter | $i = i/2 = \underbrace{[N/8]}_{2} = N/16 \rightarrow N/2^4$ |

Obs2: After k iter $i = N/2^k$

Assume: Say after k iterations code stops.

Obs1: $i = 1$ : Code Stops

Obs2: $i = N/2^k$ :

$N/2^k = 1 \iff N = 2^k \iff k = \log_2^N$

$a = b$, $a = c$ $b = c$

Use this

After $\log_2^N$ iterations Code stops.

Trace: $i = N \rightarrow N/2 \rightarrow N/4 \rightarrow N/8 \rightarrow \rightarrow 1$ : $\log_2^N$ iterations

```
void fun(int N){
    int i = N
    while(i > 0){
        print("hi")
        i = i/2
}
```

Examples:

$i = 10 \xrightarrow{i/2} 5 \xrightarrow{i/2} 2 \xrightarrow{i/2} 1 \xrightarrow{i/2} 0$ (i)

$i = 25 \xrightarrow{i/2} 12 \xrightarrow{i/2} 6 \xrightarrow{i/2} 3 \xrightarrow{3/2} 1 \xrightarrow{i/2} 0$ (i)

Trace: $i = N \rightarrow N/2 \rightarrow N/4 \rightarrow N/8 \rightarrow \rightarrow \rightarrow 1 \rightarrow 0$

$\log_2^N$ iteration + 1 iteration

Iterations = $\log_2^N + 1 = O(\log_2^N)$

```
void fun(int N){ // N>0
    for(int i=0; i<=N; i=i*2){
        print(Hi)
    }
}
```

10:25 → 10:35

$i = 0 \xrightarrow{*2} 0 \xrightarrow{*2} 0 \xrightarrow{*2} 0 \xrightarrow{*2} 0 \longrightarrow 0 \longrightarrow \infty$ loop.

```
void fun(int N){ // N>0
    for(int i=1; i<N; i=i*2){
        print(Hi)
    }
}
```

Obs1: When $i = N$ code stops

| Iterations | Initially $i = 1$ |
|---|---|
| After 1 iter | $i = i*2$ $\quad i = 2 \rightarrow 2^1$ |
| After 2 iter | $i = i*2$ $\quad i = 4 \rightarrow 2^2$ |
| After 3 iter | $i = i*2$ $\quad i = 8 \rightarrow 2^3$ |
| After 4 iter | $i = i*2$ $\quad i = 16 \rightarrow 2^4$ |

Obs2: After k iterations $i = 2^k$

Ass: Say after k iterations code stops

Obs1: $i = N$

Obs2: $i = 2^k$

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

After $\log_2 N$ iterations code stops.

```
void fun(int N){
    for(int i=1; i<=3; i++){
        for(int j=1; j<=4; j++){
            print("hi")
        }
    }
}
```

Nested loop Construct Table

| i | j | iterations |
|---|---|---|
| 1 | j: [1..4] | 4 hi |
| 2 | j: [1..4] | 4 hi |
| 3 | j: [1..4] | 4 hi |
| 4 | Stop. | |

12 hi : 12 iterations

```
void fun(int N){
    for(int i=1; i<=10; i++){
        for(int j=1; j<=N; j++){
            print("hi")
        }
    }
}
```

| i | j : [1.. N] | iterations |
|---|---|---|
| 1 | j: [1.. N] | N |
| 2 | j: [1.. N] | N |
| 3 | j: [1.. N] | N |
| ⋮ | | ⋮ |
| 10 | j: [1.. N] | N |
| 11 | Stop | |

10N iterations

```
void func(int N){
    for(int i=1; i<=N; i++){
        for(int j=1; j<=N; j++){
            print("hi")
        }
    }
}
```

| i | j : [1.. N] | iterations |
|---|---|---|
| 1 | j: [1.. N] | N |
| 2 | j: [1.. N] | N |
| 3 | j: [1.. N] | N |
| ⋮ | | ⋮ |
| N | j: [1.. N] | N |
| N+1 | Stop | |

$N*N = N^2$ iterations

```
void funcC(int N){ Q18
    for(int i=1; i<=N; i++){
        for(int j=1; j<N; j=j*2){
            print("hi")
        }
    }
}
```

| i | j=1; j<N; j=j*2 | iterations |
|---|---|---|
| 1 | j: [1.. <N] j=j*2 | $\log_2 N$ |
| 2 | j: [1.. <N] j=j*2 | $\log_2 N$ |
| 3 | j: [1.. <N] j=j*2 | $\log_2 N$ |
| ⋮ | | |
| N | j: [1.. <N] j=j*2 | $\log_2 N$ |
| N+1 | Stop: | Total iterations = $N \log_2 N$ |

```
void funcC(int N){
    for(int i=0; i<N; i++){
        for(int j=0; j<=i; j++){
            print("hi")
        }
    }
}
```

| i | j=0; j<=i; j++) [0 i] | iterations |
|---|---|---|
| | a b b-a+1 | |
| 0 | j=[0 0] = 0-0+1 | 1 |
| 1 | j=[0 1] = 1-0+1 | 2 |
| 2 | j=[0 2] = 2-0+1 | 3 |
| ⋮ | | |
| | a b b-a+1 | |
| N-1 | j=[0 N-1] = N-1-0+1 | N |

Total iterations = $1+2+3+4+\dots N = (N)(N+1)/2 = \frac{N^2+N}{2} = \frac{N^2}{2} + \frac{N}{2} = O(N^2)$

```
void funcC(int N){ TODO
    for(int i=1; i<N; i++){
        for(int j=1; j<=i; j++){
            print("hi")
        }
    }
}
```

| i | j | iterations |
|---|---|---|
| | | |

```
void func(int N){
    for(int i=1; i<N; i++){
        for(int j=1; j<=2^i; j++){
            print("hi")
        }
    }
}
```

| i | j=1; j<=2^i; j++ | iterations |
|---|---|---|
| 1 | j=1; j<=2^1; j++ : [1 2^1] | 2^1 |
| 2 | j=1; j<=2^2; j++ : [1 2^2] | 2^2 |
| 3 | j=1; j<=2^3; j++ : [1 2^3] | 2^3 |
| 4 | j=1; j<=2^4; j++ : [1 2^4] | 2^4 |
| : | | : |
| N-1 | j=1; j<=2^(N-1); j++ : [1 2^(N-1)] | 2^(N-1) |
| N | Stop. | |

Total iterations = $2^1 + 2^2 + 2^3 + 2^4 + \dots 2^{N-1}$ // G.P $a = 2, r = 2, t = N-1$

Sum of T terms G.P = $a * \frac{[r^t - 1]}{r-1} = \frac{2 * [2^{N-1} - 1]}{1} = 2 * 2^{1+N-1} - 2 = 2^N - 2 = O(2^N)$

Comparing functions : For large N value, Below will hold.

logarithmic — linear — linearlogarithmic — Polynomial — Quadratic — Exponential

$\log_2 N < \sqrt{N} < N < N\log_2 N < N\sqrt{N} < N^2 < 2^N$

| $\log_2 10000$ | $\sqrt{10000}$ | $10^4$ | $\log_2 10000 * 10000$ | $10^4 * \sqrt{10000}$ | $10^4 * 10^4$ | $2^{10000}$ |
|---|---|---|---|---|---|---|
| $\approx 13$ | 100 | $10^4$ | $13 * 10^4$ | $10^6$ | $10^8$ | $\approx 10^8$ |

Q) Higher order Term : For same N value, Term with higher value is considered higher order term

We generally do comparison for large N value.

$N >= 10^2$ Large //

a) $\log_2 N$ b) $\sqrt{N}$

a) $N\sqrt{N}$ b) $N^2$

a) $2^N$ b) $N^2$

Big(O) : What? Why? How?

O( )

How to Calculate Big O?

a) Calculate no: of iterations ✓

b) Consider only higher order Term ✓

Neglect constant coefficients ✓

| Big(O) | Calculate iterations | higher order term | Big (O) |
|---|---|---|---|
| Eg1: | $3N^2 + 5N + 10^2$ = | $3N^2$ | $\longrightarrow O(N^2)$ |
| Eg2: | $5N^2 + 10N^3 + 6N\log N$ = | $10N^3$ | $\longrightarrow O(N^3)$ |
| Eg3: | $4N^2 + 3N + 10^6$ = | $4N^2$ | $\longrightarrow O(N^2)$ |
| Eg4: | $4N + 3N\log N$ = | $3N\log N$ | $\longrightarrow O(N\log N)$ |

Eg5:
```
void fun(int N){
    i = 1; i<=10; i++){
        print("Hi"
    }
}
```
= 10 iterations → O(1) // Indicates fixed no: of iterations.

↓ Not depending on N value: Fixed / Constant

Eg5:
```
void fun(int N){
    i = 1; i<=100; i++){
        print("Hi"
    }
}
```
= 100 iterations → O(1)

↓ Fixed iterations

logarithmic

$\log_2 N$

linear

$N$

linearlogarithmic

$N \log_2 N$

Quadratic

$N^2$

Exponential

$2^N$

Polynomial

$\sqrt{N}$, $N\sqrt{N}$