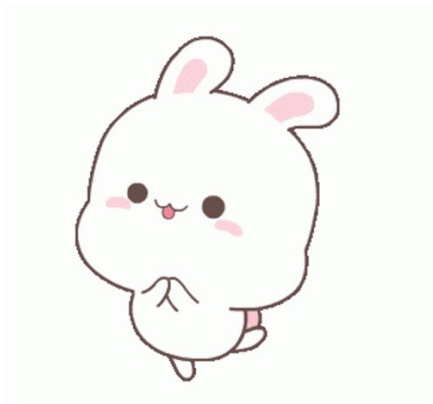

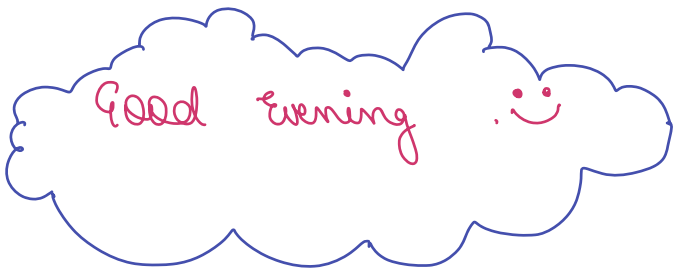

Today's Content

log Basics
Calculating iterations in given codes:
Intro to BigO on how to calculate it.
TLE. (Time limit exceeded)?
Importance of constraints

→ log is inverse of exponential function

Basics of log :-

$\log_b a$ = To what value we have to raise b to get a

Q1. $\log_2 64 = ?$ $\quad a=2^4$, $b=2$, $\quad 2^6 = 64$, ans = 6.

Q2. $\log_5 25 = ?$ $\quad 5^2 = 25$, ans = 2.

Q3 $\log_2 32 = 5 \longleftrightarrow 2^c = 2^5 \Rightarrow \boxed{c=5}$

Q4. $\log_2 16 = 4$, $2^4 = 16$, ans = 4.

Q4. $\log_2 10 = \underbrace{3}_{\text{Integral part (floor value)}}$

$2^3 = 8$

$2^4 = 16$

Q5. $\log_2 40 = 5$

$2^4 = 16$

$2^5 = 32$

$2^6 = 64$

1) $\log_a a^x = x$

Q- Given a positive integer N, how many times we need to divide it by 2 until it reaches 1.

N = 100.

$100 \rightarrow 50 \rightarrow 25 \rightarrow 12 \rightarrow 6 \rightarrow 3 \rightarrow 1$

↳ 6 times

Quiz1: How many times we need to divide 9 by 2 till it reaches 1.

$9 \to \frac{9}{2} \to \frac{9}{4} \to \frac{9}{8}$

$9 \to 4 \to 2 \to 1 \Rightarrow$ 3 times.

$\log_2 16 = 4$ (a) (b) (c)

$16 = 2^4$.

Generalize:

$N \to \frac{N}{2} \to \frac{N}{2^2} \to \frac{N}{23} \text{ - - - } 1$

No. of terms = K.

$\frac{N}{2^K} = 1$

$\log_2 N = K$.

$N \to \frac{N}{2^0} \to \frac{N}{2^1} \to \frac{N}{2^2} \to \frac{N}{2^3} \text{ - - - } \to \frac{N}{2^K}$

$N = 2^K$

$\log_b a = c$

$a = b^c$

2). if $N = 2^K$, $\log_2 N = K$.

Quiz2: How many times we need to divide 27 by 2 till it reaches 1.

$N = 27$.

$\frac{27}{2} \to \frac{13}{2} \to \frac{6}{2} \to \frac{3}{2} \to 1 \Rightarrow$ 4 steps.

No. of steps.

$K = \log_2 N$

$= \log_2 27 \Rightarrow 4$.

$2^4 = 16$

$2^5 = 32$

Iterations:

$i = 17, \log_2 17 \Rightarrow 4$

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

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

$i = 17, \xrightarrow{i/2} 8 \xrightarrow{i/2} 4 \xrightarrow{i/2} 2 \xrightarrow{i/2} 1$ (stop).

Obs1: when $i=1$, stop.

$N \to \frac{N}{2} \to \frac{N}{2^2} \to \frac{N}{2^3} \cdots 1$ (stop)

No. of iterations $k = \log_2 N$ iteration

$i = i/3$.

$N \to \frac{N}{3} \to \frac{N}{3^2} \to \frac{N}{3^3} \Rightarrow \log_3 N$

Iterations	initially $i = N$.
After 1 iteration.	$i = i/2 \Rightarrow \frac{N}{2} \Rightarrow N/2^1$
After 2 iteration	$i = i/2 \Rightarrow \frac{\lfloor N/2 \rfloor}{2} \Rightarrow \frac{N}{4} \Rightarrow N/2^2$
After 3 iteration	$i = i/2 \Rightarrow \frac{\lfloor N/4 \rfloor}{2} \Rightarrow \frac{N}{8} \Rightarrow N/2^3$

Obs2: After K iterations, $i = \frac{N}{2^K}$

Obs1: $i = 1$, code stop.

Obs2: $i = \frac{N}{2^K}$ $\quad \frac{N}{2^K} = 1$

$N = 2^K$

$K = \log_2 N$

$i: N \to \frac{N}{2} \to \frac{N}{4} \to \frac{N}{8} \cdots 1 \Rightarrow \log_2 N$ iterations.

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

$\log_2 N$ iterations

$N=32 \xrightarrow{\div 2} 16 \xrightarrow{\div 2} 8 \xrightarrow{\div 2} 4 \xrightarrow{\div 2} 2 \xrightarrow{\div 2} 1$

$32 \xleftarrow{\times 2} 16 \xleftarrow{\times 2} 8 \xleftarrow{\times 2} 4 \xleftarrow{\times 2} 2 \xleftarrow{\times 2} 1$

Obs1: when $i=N$, code stop.

Iterations	initially, $i=1$.
After 1 iteration	$i=i*2 = 2 = 2^1$
After 2 iteration	$i=i*2 = 2*2=4 = 2^2$
After 3 iteration	$i=i*2 = 4*2=8 = 2^3$

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

$i=N$

$i=2^k$, $N=2^k$

$k = \log_2 N$

After $\log_2 N$ iterations, code stop

void fun(int N){ // N>0.

```
for(int i = 1; i <= N ; i = i*2){
    print("Hi");
}
```

}

$N = 16$.

$i = 1 \xrightarrow{*2} 2 \xrightarrow{*2} 4 \xrightarrow{*2} 8 \xrightarrow{*2} 16 \xrightarrow{*2} 32$ [stop]

$\log_2 17 \Rightarrow 4$

Obs1: when $i = N+1$, code stops.

Obs2: After K iterations, $i = 2^K$

$$2^K = N+1$$

$$K = \log_2(N+1)$$

After $\log_2(N+1)$ iterations, code stops

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

i = 0,
i = i*2
i = 0*2 ⇒ 0
# infinite times

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

i	j	iterations
1	j:[1--N]	N, Hi
2	j:[1--N]	N, Hi
3	j:[1--N]	N, Hi
⋮		
10	j:[1--N]	N, Hi
11	stop.	N+N--10
		N*10

10N iterations

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

i	j	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

```
void fun(int N){
    for(int i=1; i<=N; i++){
        for(int j=1; j<=N; j++){
            break;
        }
    }
}
```

i	j	iteration
1	j:[1--N]	1
2	j:[1--N]	1
⋮		
N	j:[1--N]	1
		1*N

N iterations

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

i	j	iterations
1	j=1; j<N; j=j*2;	logN
2	j=1	logN
3	j=1; j<N; j=j*2	logN
⋮	j=1; j<N; j=j*2	
N	j:	logN
		N* logN

NlogN iterations

$O(N \log_2 N)$

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

i	j	iterations
1	[1 1]	1
2	[1 2]	2
3	[1 3]	3
4	[1 4]	4
5	[stop]	

Sum of 1st 4 natural no. = 10

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

i	j	iterations
1	[1..1]	1
2	[1..2]	2
3	[1..3]	3
⋮		⋮
N	[1..N]	N

Sum of 1st N terms = $\frac{N(N+1)}{2}$

$N*(N+1)/2$ iterations

$\frac{N^2+N}{2} \Rightarrow O(N^2)$

```
void fun(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++ . iteration
1	[1 -- 2^1] 2^1
2	[1 -- 2^2] 4 = 2^2
3	[1 -- 2^3] 2^3
4	[1 -- 2^4] 2^4.
⋮	
N-1	[1 -- 2^(N-1)] 2^(N-1)
N	[1 -- 2^N] 2^N.

Total iteration $= 2^1 + 2^2 + 2^3 \cdots 2^N$

$\frac{a^m}{a^n} \Rightarrow a^{m-n}$

$a = 2^1$, $r = \frac{2^2}{2^1} = 2, \frac{2^3}{2^2} \Rightarrow 2.$

$r = 2.$

, $t = N$

Sum of (N) terms of GP.

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

$$= \frac{2(2^N - 1)}{2-1} \Rightarrow 2(2^N - 1) \text{ iterations}$$

$a^m * a^n \Rightarrow a^{m+n}$

Comparing Iterations:-

Q→ Say for a Question, following codes are submitted:-

	Harshikesh (Algo1)	Utkarsh (Algo2)
iterations :	$100 \log_2 N$	$N/10$.

$N = 3550$.

(For small inputs)

$N < 3550$, Algo2 $(\frac{N}{10})$ is better

$N \geq 3550$, Algo1 $(100 \log_2 N)$ is better.

(For large inputs)

1 Billion → 2 Billion → 3 Billion

Baby Shark video → 3 Billion

Ind Vs Pak ⟶ 5 crore.

Real world → larger inputs ∴ algo1 is better.

Asymptotic Analysis of Algorithms:-

Analyzing performance of algorithm for very large inputs

3 Notations:

1. Big O 2. Omega 3. Theta.

How to calculate Big O.

1. Calculate no. of iterations
2. Consider only Higher Order terms & neglect constant coefficients

Comparing functions.

$\log N < \sqrt{N} < N < N\log N < N\sqrt{N} < N^2 < N^3 < 2^N < N! < N^N$

$N = 36$.

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

Big(O). Calculate iterations. Higher order term. Big(O)

Ex1: $3N^2 + 5N + 10^2 = 3N^2$ $O(N^2)$

Ex2: $5N^2 + 10N^3 + 6N\log N = 10N^3$ $O(N^3)$

Ex3: $4N^2 + 3N + 10^6 = 4N^2$ $O(N^2)$.

Quiz12: $4N + 3N\log N + 1 = 3N\log N$ $O(N\log N)$.

Quiz13: $4N\log N + 3N\sqrt{N} + 10^6 = 3N\sqrt{N}$ $O(N\sqrt{N})$

$(a^m)^n \Rightarrow a^{mn}$

Ex5:
```
void fun(int N){
    for(int i=1; i<=10; i++){
        print("Hi");
    }
}
```
= 10 iterations ⇒ O(1)

↓

Not depending on N.

Fixed / Constant

Ex6:
```
void fun(int N){
    for(int i=1; i<=100; i++){
        print("Hi");
    }
}
```
= 100 iteration ⇒ O(1).

↓

Fixed iteration

TODO: Calculate O() for all above examples

Q). Why consider only higher order term?

Say $f(n) = N^2 + 10N$ / Iterations higher order: N^2 lower order: $10N$

Input size	Total iterations	% lower order term's iteration in total
N = 10	$f(10) = 10^2 + 10*10 = 200$	$\frac{10*10}{200} \times 100 = 50\%$
N = 100	$f(100) = (10^2)^2 + 10\times100 = 10^4 + 10^3$	$\frac{10^3}{10^4+10^3} \times 100 = \frac{10^5}{10^4+10^3} \Rightarrow 10\%$
$N = 10^4$	$f(10^4) = (10^4)^2 + 10\times10^4 = 10^8 + 10^5$	$\frac{10^5}{10^8+10^5} \times 100 = \frac{10^7}{10^8+10^5} \Rightarrow 0.1\%$

Obs: As input increases, contribution of lower term decreases, negligible.

Neglect constant coefficients:

	Algo1	Algo2.	iterations	code faster
Q1.	$10 \log_2 N$	N^2	N	$10 \log_2 N$
Q2.	$100 \log_2 N$	N	⟶	$100 \log_2 N$
Q3.	$10^3 \log_2 N$.	$N/10$	⟶	$10^3 \log_2 N$
Q4.	$10N$	$N^2/10$.	$\frac{N^2}{10}$ ⟶	$10N$

Issues in big O.

I:

Algo1 $10^3 N$ Algo2. N^2 less iterations code faster.

Big O: → larger inputs. (smaller inputs fail)

Ex.

N	Algo1 $10^3 N$	Algo2. N^2	$N < 10^3$
Big O:	$O(N)$	$O(N^2)$	faster algo is.
10	10^4.	10^2	algo2 is faster.
10^2	10^5	10^4	algo2 is faster.
10^3	10^6	$(10^3)^2 = 10^6$	same.
10^3+1	$10^3 * (10^3+1)$	$(10^3+1)(10^3+1)$	algo1 is faster.
10^4	$10^3 * 10^4 = 10^7$	$(10^4)^2 = 10^8$	algo1 is faster

II

	Algo1	Algo2	less iterations	faster
	$10N^2$	$2N^2+5N$	$10N^2$: more iterations	algo2 is faster.
big O:	$O(N^2)$	$O(N^2)$.		

Note: Algo1 is same as Algo2.

Imp. Code. Search for K in ar[].

```
boolean search(int ar[], int K){
    int n = ar.length;
    for(int i=0; i<n; i++){
        if(ar[i] == K){
            return true;
        }
    }
    return false;
}
```

Best	Worst
K → ar[0]	K ≠ ar[i]
1 iteration	N iterations

Note: While calculating we consider worst case scenario