17/07/25

DSA
Time Complexity

★ Basics of Logarithm:

What is the meaning of Log?
It is the reverse of exponent function.

$$\log_b(a) = c$$
$$b^c = a$$

eg- $\log_2 64 = ?$

$b = 2 \quad a = 64$

$2^c = 64$
$2^6 = 64$
$\boxed{c = 6}$

Now, calculate the floor values of the following logarithms

$\log_2 10 \Rightarrow 2^c = 10 \Rightarrow 2^3 \simeq 10 \Rightarrow 3$

$\log_2 40 \Rightarrow 2^c = 40 \Rightarrow 2^5 \simeq 40 \Rightarrow 5$

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

$2^k = 2^6$
$\boxed{k = 6}$

$\log_3(3^5)$
$3^k = 3^5$
$\boxed{k = 5}$

$$\boxed{\log_a(a^n) = n}$$

(Q) How many times we need to divide N by 2 till it reaches 1 ?

N = 100

100 → 50 → 25 → 12 → 6 → 3 → 1 : 6 times

Consider floor values

9/2 → 4/2 → 2/2 → 1 : 3 times

Observation :

$N/1 \rightarrow N/2 \rightarrow N/4 \rightarrow N/8 \rightarrow N/16 \ldots \rightarrow 1$

$N/2^0 \rightarrow N/2^1 \rightarrow N/2^2 \rightarrow N/2^3 \rightarrow N/2^4 \ldots \rightarrow N/2^k$

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

Now, we will try to apply the approach →

27 → 13 → 6 → 3 → 1 : 4

$N/2^0 \rightarrow N/2^1 \rightarrow N/2^2 \rightarrow N/2^3 \rightarrow N/2^4$

$\simeq$

$k = \log_2 N \Rightarrow \log_2 27 \Rightarrow \log_2 (2^4)$

$\therefore \boxed{k = 4}$

We consider floor value

Q) How many iterations will be there in this loop?

⇒
```
N=D, i = N;
while (i>1) {
    i=i/2;
}
```

Iteration	Initializing of i=N
After 1st	$i = i/2 \rightarrow i = N \Rightarrow i = N/2 = N/2^1$
2nd	$i = i/2 \rightarrow i = N/2 \Rightarrow i = N/4 = N/2^2$
3rd	$i = i/2 \rightarrow i = N/4 \Rightarrow j = N/8 = N/2^3$
4th	$i = i/2 \rightarrow i = N/8 \Rightarrow i = N/16 = N/2^4$

After k no. of iterations

k= iterations

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

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

$N = 2^k$

Take log both sides

$\boxed{k = \log_2 N}$

So, whenever your number is being used in the loop, that number is either getting divide or multiplied by 2, in both cases the no. of iterations will be $\boxed{\log_2 N}$.

i = 1

i*2 → 2*2 → 4*2 → 8*2 → 16 ....

2^0 2^1 2^2 2^3 2^4 ....

$\therefore k^{th}$ itr $= 2^{k-1} = N-1$

Take log on both sides

$k-1 = \log_2(N) - 1$

(No. of iterations) $k = \log_2 N$

★ Nested loops:

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

value of i	Range of j	Iteration in j
1	j: [1 N]	N
2	j: [1 N]	N
3	j: [1 N]	N
⋮		
10	j: [1 N]	N

Total iterations = N+N+N+N+N+N+N+N+N+N

= 10 N

Q) for (i → 1 to 4): {
    for (j → 1 to i) {
        // print (i+j)
    }
}

i	range(j)	Iteration(j)
1	j = [1 1]	1
2	j = [1 2]	2
3	j = [1 3]	3
4	j = [1 4]	4
	→ exit	

Total iterations ⇒ 1+2+3+4
= 10

sum of ~~n~~ natural numbers $\frac{n*(n+1)}{2}$

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

i	range(j)	Iteration(j)
1	j: [1 2^1]	2^1
2	j: [1 2^2]	2^2
3	j: [1 2^3]	2^3
.		.
.		.
N	j: [1 2^N]	2^N

Total iterations = $2^1+2^2+2^3+\ldots+2^N$

Sum of GP $= \frac{a(r^n - 1)}{(r-1)}$; $a=2, r=2, n=n$

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

$= 2 * (2^n - 1)$

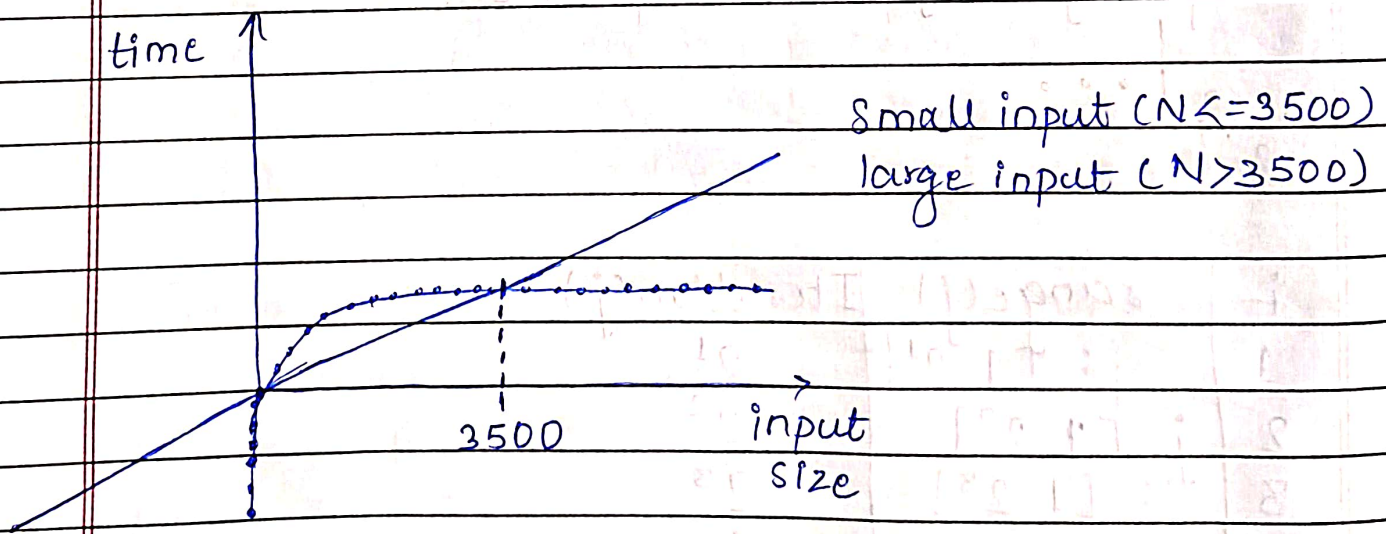
★ Comparing iterations of two algorithms using Graphs:

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

$N = 36$

$5 < 6 < 36 < 180 < 216 < 1260 < 2^{36}$

	Algo 1 (·····)	Algo 2 (——)
Iterations	$100 * \log_2(N)$	$N/10$



So, time taken by Algo 1 until 3500 is ~~less~~ more than Algo 2.

Now, $N = 10000$

(2) $\frac{N}{10} = 100$

(1) $100 * \log_2(N)$

✱ Asymptotic Analysis of Algorithm :

Asymptotic Analysis or Big(O) means analyzing performance of algorithm for larges inputs.

- Steps of calculation of Big (O):
- Calculate iterations based on input size
- Ignore lower order terms
- Ignore constant coefficients.

$100 * \log_2(N) \rightarrow 100 \rightarrow$ constant (discard)

$O(\log_2 N)$

$N/10 \rightarrow 1/10 \rightarrow$ constant (discard)

$O(N)$

✱ Calculate the Big(O) from the number of iterations :

eg ⇒ $f(n) = 4n^2 + 3n + 1$

$= 4n^2$ (highest order)

$= n^2$ ($4 \rightarrow$ constant $\rightarrow$ discard)

$= O(n^2)$

☆ Why are we considering higher order term:

iteration = $n^2 + 10n$

↗ high order ↑ lower order

Value of n	Iteration count (n^2+10n)	% of lower order term in total iterations
10	$100+100$	$\frac{100}{100+100} \times 100 = 50\%$
100	10^4+10^3	$\frac{10^3}{10^4+10^3} \times 100 = 10\%$
1000	10^6+10^4	$\frac{10^4}{10^6+10^4} \times 100 = 1\%$

Conclusion: For higher value of n, contribution of lower order term is very less in total itr, that is why we can discard lower order term.

★ Why do we neglect constant coefficient in Big(O)

	Algo 1	Algo 2
	n^2	$5n$
TC:	$O(n^2)$	$O(n)$

Algorithms	$3n^2$	$7n^2$	$6n^2$
$n=1$	3	7	6
$n=100$	30000	70000	60000
$n=10000$	3×10^8	7×10^8	6×10^8

Since $3n^2$, $7n^2$ & $6n^2$ are equivalent whe we deal with large value of n.

→ we can discard constant coefficient for any term in Big O notation.

★ Issues in Big O Notation :

(1)

	Algo 1	Algo 2
itr	$10n$	n^2
Big(O)	$O(n)$	$O(n^2)$

Best algo. acc to Big O is $O(n)$

Claim: $O(n)$ is better than $O(n^2)$
Is it always true?

	$10n$	n^2	
Value of n	Itr in Algo1	Algo2	Which is better
1	10	1	Algo 2
5	50	25	Algo 2
8	80	64	Algo 2
10	100	100	Same
11	110	121	Algo1
12	120	144	Algo 1

Issue: for $n >= 11$ Algo 1 is better

Algo 1 is better than algo2 after some certain value [threshold value]

(2) 1. for (i=1; i<=N; i++) {
    |
}
[1 N], step ⇒ 1
N itr
O(N)

2. for (i=1; i<=N; i=i+2) {
    |
}
[1 N], step = +2
N/2 itr
O(N)

But Algo 2 is better, still the iterations as checked by Big (O) is same.

Big O has its own issues, but in general for larger values it is considered as best.

★ Time Limit Exceeded error :

$1 <= N <= 10^8$

Assumption we know, 1 sec → 10^8 itr

N iteration
(10^8 itr)

But, N^2 iteration we have as we wrote the code that way.

$(10^8)^2 \Rightarrow 10^6 > 10^8$ Not possible

$1 <= N <= 10^{10}$

N itr No ✗

$\sqrt{N}$ itr Yes ✓