Time Complexity

→ Log Basics + Iterations

01. Comparing algos
   ↳ Using iterations & graphs

02. Big O
   (a) why lower order terms are neglected
   (b) why constant coeff are neglected
   (c) Issues in Big O
   (d) Worst case

03. TLE
   (a) why TLE occurs & info about online editors
   (b) Importance of constraints

Log → Log is inverse of exponential function

$\log_b a$ ⇒ To what value we have to raise b, such that we get a.

$$\log_b a = c \longleftrightarrow b^c = a$$

01. $\log_2 16 = 4 \longleftrightarrow 2^c = 16$

$2^c = 2^4$

$c = 4$

02 $\log_3 27 = 3$

03. $\log_2 2^5 = 5 \longleftrightarrow 2^c = 2^5$

$c = 5$

$$\log_a a^x = x$$

04. $\log_2 10 = 3 \longleftrightarrow 2^c = 10$

Integer part of c

$2^c = 2^{3....}$

$c = 3 \ldots$

05. $\log_3 30 = 3$

06 How many times do we need to divide 100 by 2 to get 1? Ans = 6

$$\frac{100}{2} = \frac{50}{2} = \frac{25}{2} = \frac{12}{2} = \frac{6}{2} = \frac{3}{2} = 1$$

07 How many times we need to divide 9 by 2 to reach at 1? Ans=3

$$\frac{9}{2} = \frac{4}{2} = \frac{2}{2} = 1$$

$$N \to \frac{N}{2} \to \frac{N}{4} \to \frac{N}{8} \ldots\ldots 1$$

After K divison

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

$$\frac{N}{2^K} = 1 \longrightarrow N = 2^K \longleftrightarrow \log_2 N = K$$

$$a = b^c \longleftrightarrow \log_b a = c$$

* How many times we need to divide 27 by 2 to reach 1

$$\log_2 27 = 4$$

No. of iterations

01.
```
i = N
while (i > 1) {
    i = i / 2
}
```

$$N \to \frac{N}{2} \to \frac{N}{4} \to \frac{N}{8} \dots 1$$

No. of iterations $= \log_2 N$

$8 \xrightarrow{\div 2} 4 \xrightarrow{\div 2} 2 \xrightarrow{\div 2} 1$

$8 \xleftarrow{*2} 4 \xleftarrow{*2} 2 \xleftarrow{*2} 1$

02.
```
for (i=1 ; i<N ; i = i * 2)
{
    ===
}
```

$i = N$
$i = 2^k$ } stopping condition

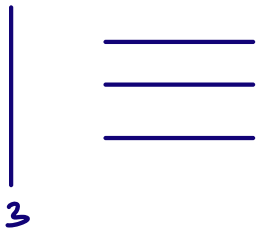
$N = 2^k$

$K = \log_2 N$

iteration	value of i after iteration
1	$2 = 2^1$
2	$4 = 2^2$
3	$8 = 2^3$
4	$16 = 2^4$
.	.
.	.
.	.
.	.
K	2^K

$N > 0$

03 `for(i=0 ; i<N ; i=i*2){`



$i = 0 * 2$

∴ infinite no. of times

04. `for( i=1 ; i≤10 ; i++){`

`for(j=1 ; j≤N ; j++){`

`SOP("work");`

3

3

i	j	No. of iteration
1	[1 N]	N
2	[1 N]	N
3	[1 N]	N
4	[1 N]	N
.	.	.
.	.	.
.	.	.
10	[1 N]	N

No. of iterations $= \underbrace{N+N+N \cdots N}_{10}$

$= 10 * N$

$= 10N$

TC : $O(N)$

```
for ( i=1 ; i ≤ N ; i++) {
    for ( j=1 ; j ≤ N ; j++) {
        SOP("work");
    }
}
```

i	j	No. of iteration
1	[1 N]	N
2	[1 N]	N
3	[1 N]	N
4	[1 N]	N
.	.	.
.	.	.
.	.	.
N	[1 N]	N

Iterations = $\underbrace{N+N+N \ldots . N}_{N \text{ times}}$

$= N^2$

07.

```
for ( i=1 ; i ≤ N ; i++) {
    for ( j=1 : j ≤ N ; j=j*2)
        SOP("work");
}
```

i	j	No. of iteration
1	[1 N]	log N
2	[1 N]	log N
3	[1 N]	log N
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
N	[1 N]	log N

No. of iterations = $N * \log N$

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

i	j	No. of iteration
1	[1 1]	1
2	[1 2]	2
3	[1 3]	3
4	[1 4]	4

No. of iterations = 1+2+3+4 = 10

Q9

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

i	j	No. of iteration
1	[1 1]	1
2	[1 2]	2
3	[1 3]	3
⋮	⋮	⋮
N	[1 N]	N

No. of iteration = $1+2+3+\cdots+N$

$$= \frac{N*(N+1)}{2} = \frac{N^2+N}{2}$$

TC : $O(N^2)$

10.
```
for (i=1 ; i≤N ; i++){
    for (j=1 ; j≤2^i ; j++){
        ==
    }
}
```
3 (inner loop) 3 (outer loop)

i	j	No. of iteration
1	[1 2]	2
2	[1 2^2]	2^2
3	[1 2^3]	2^3
4	[1 2^4]	2^4
⋮	⋮	⋮
N	[1 2^N]	2^N

Total $= 2^1 + 2^2 + 2^3 + 2^4 + \cdots + 2^N$

$a = 2^1$

$r = 2$

No. of terms $= N$

Sum of GP $= \dfrac{a(r^n - 1)}{r - 1}$

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

Total iterations $= 2(2^n - 1)$

Comparing two Algorithms → we need to go with no. of iterations

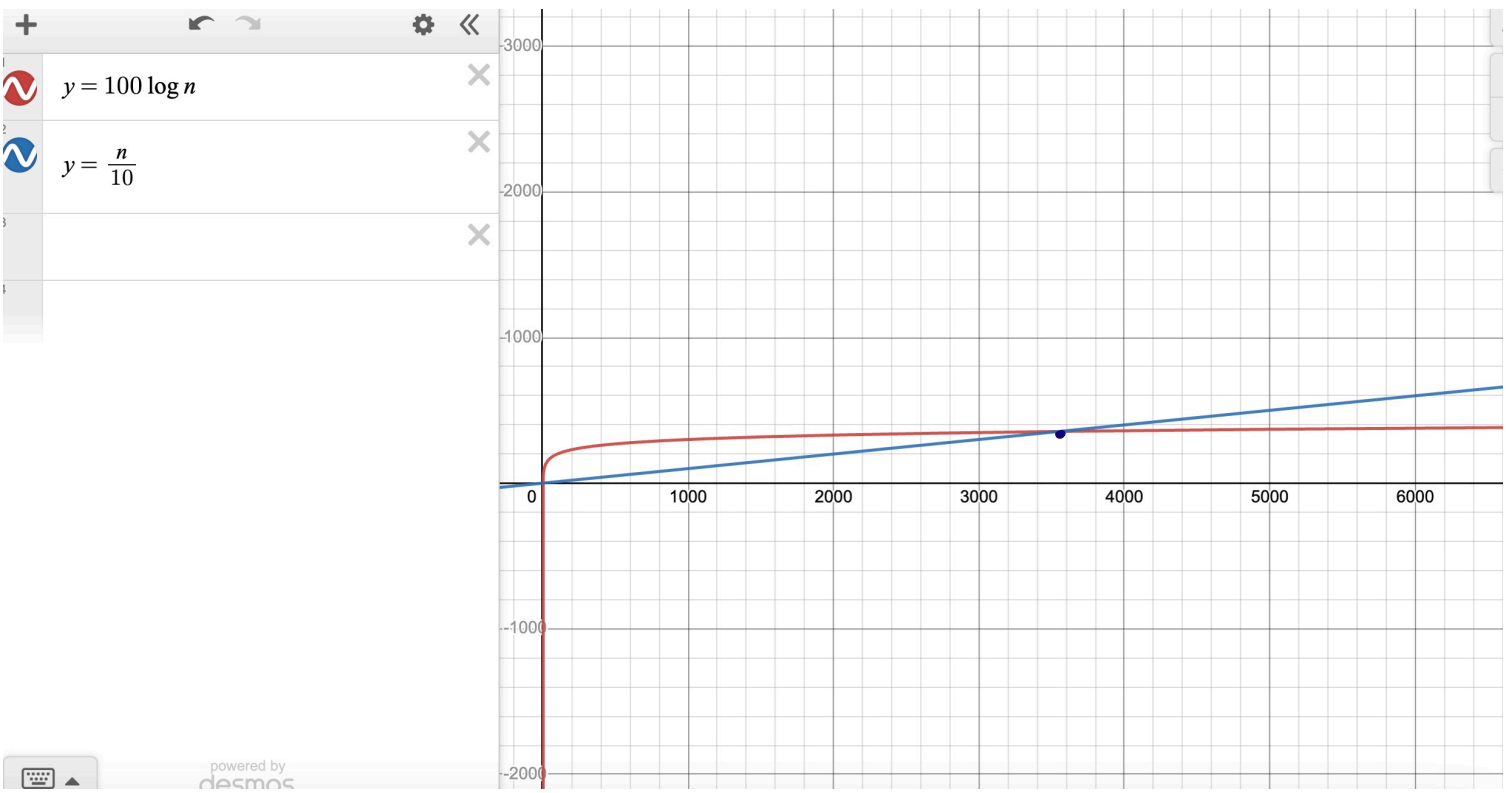
Compare two Algo on basis of no. of iteration

	Algo 1	Algo 2
Iteration →	$100 \log(N)$	$N/10$

$N = 3550$

$N < 3550$ Algo 2 ($N/10$) is better

$N \geq 3550$ Algo 1 ($100 \log N$) is better



Baby shark video → 13 billion

Ind vs Pak (T20 world cup) → 1.8 crore

Ind vs Pak (Asia cup) → 4 crore

Ind vs Pak (World cup) → 5.6 crore

Since we are dealing with huge data, largest input in our real world, so we should compare Algo taking very large Input

Conclusion → Algo 1 is better since it is having less iteration for larger input

Comparison order in terms of iterations

$K < \log(N) < Sqrt(N) < N < N\log N < N\,Sqrt(N) < N^2 < N^3 .. 2^n$

↓ constant

→ Increasing no. of iterations

log(N) is Best TC

2^n is worst TC

8:04 AM → 8:14 AM

Asymptotic Analysis / Big O

↳ Analysing the performance of Algo for larger inputs

* Steps to calculate Big O

01. Calculate No. of iterations wrt input
02. Ignore the lower order terms
03. Ignore the constant coefficient.

Eg :- Calculate Big(O)

01. $100 \log N \longrightarrow O(\log N)$

02. $\frac{N}{10} \longrightarrow O(N)$

03. $4N^2 + 10N \longrightarrow O(N^2)$

04. $4N + 3N\log(N) + 1 \longrightarrow O(N\log(N))$

05. $4N\log N + 3N\,\text{sqrt}(N) + 10^6 \longrightarrow O(N\,\text{sqrt}(N))$

06. $10^6 \longrightarrow O(1)$

$N = 10^6$

$4 * 10^6 * \underbrace{\log 10^6}_{6} + 3\,10^6 + \underbrace{\text{sqrt}(10^6)}_{10^3}$

Ignore the lower order terms?

Algo → $N^2 + 10N$ iterations

N	Total iteration	Lower order term = 10N	% of contribution of lower order
10	200	100	$\frac{100}{200} * 100 = 50\%$
100	11,000	1000	$\frac{1000}{11,000} * 100 = 9\%$
10^4	$(10^4)^2 + 10 * 10^4$ $= 10^8 + 10^5$	$10 * 10^4$ 10^5	$\frac{10^5}{10^8 + 10^5} * 100 = 0.1\%$

Observations → As we increase our input size, the contribution of lower order terms becomes very insignificant, so ignore it.

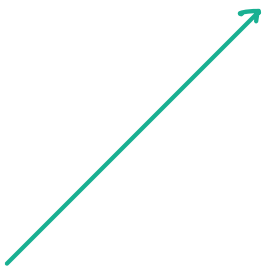
Neglect constant coefficients

Big O → Rate of growth of function w.r.t larger inputs

$f(y) = x$ ⟶ linear

$f(y) = 10x$ ⟶ linear

$f(y) = 100x$ ⟶ linear



Algo 1	Algo 2	Better
$10 \log N$	N	Algo 1
$100 \log N$	N	Algo 1
$10 N$	N^2	Algo 1

Issues → What if the constant part is very large?

Issues with Big O

01. Algo 1 $\longrightarrow$ $10^3 N$ $\longrightarrow$ $O(N)$

02 Algo 2 $\longrightarrow$ N^2 $\longrightarrow$ $O(N^2)$

Algo 1 is better than Algo 2

Input Size	Algo 1	Algo 2
10	10^4	10^2
100	10^5	10^4
1000	10^6	10^6
1000+1	$10^3(10^3+1)$	$(10^3+1)(10^3+1)$
10^4	10^7	10^8

Issue 1 = Big O only gives correct answer after a certain point.

Issue 2

Algo 1 $\rightarrow$ $10N^2$ $\longrightarrow$ $O(N^2)$

Algo 2 $\rightarrow$ $1000N^2$ $\longrightarrow$ $O(N^2)$

Algo 1 is same as Algo 2

Q Calculate the no. of odd values from 1 to 10.

→
```
for(i=1; i≤10; i++){
    if(i%2 != 0){
        ans++;
    }
}
```

No. of iteration = N

TC: O(N)

→
```
for(i=1; i≤10; i=i+2){
    ans++;
}
```

No. of iterations = $N/2$

TC: O(N)

Issue 2 = When higher order terms are same for two Algos, Big O can't judge properly.

* Time Limit Exceeded Error (TLE)

Yogesh

Amazon ⟶ Idea ⟶ Implemented ⟶ TLE

Idea2 ⟶ Implement ⟶ TLE

Q Is it really necessary to implement the idea to check if it is going to work? ⇒ NO

Without even writing a single line of code, you should be able to judge if it is going to work.

Working of online Editors

Online Editor —Run→ Online servers

→ Processing speed = 1 GHz

→ 10^9 instructions per second

Code must be executed in 1 sec = 10^9 instructions

```
int countfactor(N)
{
    int ans = 0
    for(i=1; i≤N; i++){
        if(N%i==0) ans++;
    }
}
```

per iteration = 7 ~ 8 instruction

declare variable
operator
calling function
if-else clauses
incrementing

1 iteration = 10 instruction

1 sec = 10^9 instructions

= 10^8 * 10 instructions

= 10^8 * 1 iteration

1 sec = 10^8 iterations

1 iteration = 100 instructions

1 sec = 10^9 instructions

= 10^7 * 100 instructions

= 10^7 * 1 iteration

1 sec = 10^7 iterations

STANDARD → 1 second = $10^7 - 10^8$ iterations

* General structure question

→ Description

→ Constraints

→ Input & Output format

→ Example

Q Given arr[N] ... do something

Constraints = $1 \le N \le 10^5$ → size of array

$1 \le A[i] \le 10^9$

Idea 1 = 3 nested loops → TC = $O(N^3)$

No. of iterations $= (10^5)^3$

$= 10^{15} \rightarrow$ TLE

Idea 2 = 2 nested loops $\rightarrow$ TC = $O(N^2)$

No. of iterations $= (10^5)^2$

$= 10^{10} \rightarrow$ TLE

Idea 3 = 1 Loop $\rightarrow$ TC = $O(N)$

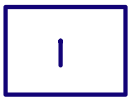
No. of iterations $= 10^5$ ✓

Q Given A[N], calculate & return sum of all A[] elements

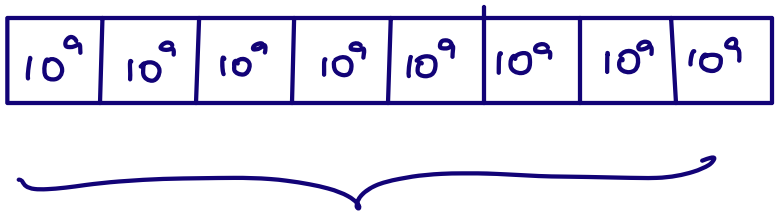
Constraints :- $1 \le N \le 10^5$

$1 \le A[i] \le 10^9$

```
int sum(int [] A)
{
    ~~int~~ long ans = 0
    for(i=0; i<N; i++){
        ans = ans + A[i];
    }
    return (int) ans;
}
```



Sum = 1



10^5 size

$$Sum = \underbrace{10^9 + 10^9 + 10^9 + \dots + 10^9}_{10^5}$$

$$Sum = 10^{14}$$

int $\leq 10^9$

long $\leq 10^{18}$

int ✗

long ✓

Doubts

Compare Two Algorithm

which one is better ? ⇒ One with less iteration

Algo 1	Algo 2	
100	10	⟶ Algo 2
$100 \log N$	$10N^2$	⟶ Algo 1
$100N^2$	$10N^3$	⟶ Algo 1