Logarithms

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

$\log_2(64) = 6$ $\qquad 2^6 = 64$

$\log_5(25) = 2$ $\qquad 5^2 = 25$

$\log_2(10) = 3.\_$ $\qquad 2^3 < 10 < 2^4$ (8 $\quad$ 16)

$\text{int}(\log_2(40)) = \underline{5}$ $\qquad 2^5 < 40 < 2^6$

Q → Given a +ve integer N, how many times we need to divide it by 2 to reach 1.

(integer division) $\frac{5}{2} = 2$

Eg → 50

$50 \longrightarrow 25 \longrightarrow 12 \longrightarrow 6 \longrightarrow 3 \longrightarrow 1$ $\qquad$ Ans = 5

$9 \longrightarrow 4 \longrightarrow 2 \longrightarrow 1$ $\qquad$ Ans = 3

$27 \longrightarrow 13 \longrightarrow 6 \longrightarrow 3 \longrightarrow 1$ $\qquad$ Ans = 4

$$N \longrightarrow \frac{N}{2} \longrightarrow \frac{N}{2^2} \longrightarrow \frac{N}{2^3} \cdots \boxed{\frac{N}{2^K} = 1}$$

→ After K steps

$$\Rightarrow N = 2^K$$

$$\Rightarrow \boxed{\log_2(N) = K}$$

No. of iterations

1) // N > 0

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

#iterations = $\log_2(N)$

TC = $O(\log(N))$

$i = N \longrightarrow \frac{N}{2} \longrightarrow \frac{N}{2^2} \ldots \frac{N}{2^k} = 1$

2)

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

$i = 1 \longrightarrow 2 \longrightarrow 2^2 \longrightarrow 2^3 \ldots N$

#iterations = $\log_2(N)$ TC = $O(\log(N))$

3) // N >= 0

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

$i = 0 * 2 = 0$

iterations = ∞

4)

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

i	j	#iterations
1	[1 N]	N
2	[1 N]	N
:		

10	[1 N]	N

10 * N ✓ TC = O(N)

5)
```
for (i = 1 ; i <= N ; i++) {
    for (j = 1 ; j <= N ; j++) {
        :
    }
}
```

i	j	#iterations
1	[1 N]	N
2	[1 N]	N
⋮		
N	[1 N]	N

N * N ✓ TC = O(N^2)

6)
```
for (i = 1 ; i <= N ; i++) {
    for (j = 1 ; j <= N ; j = j*2) {
        :
    }
}
```

i	j	#iterations
1	1 _ N	$\log_2(N)$
2	1 _ N	$\log_2(N)$
⋮		⋮
N	1 _ N	$\log_2(N)$

N * $\log_2(N)$ TC = O(N log(N))

7)
```
for (i = 1 ; i <= 4 ; i++) {
    for (j = 1 ; j <= i ; j++) {
        :
    }
}
```

i	j	#iterations
1	[1 1]	1
2	[1 2]	2
3	[1 3]	3
4	[1 4]	4
		10 ✓

TC = O(1)

8)
```
for (i = 1 ; i <= N ; i++) {
    for (j = 1 ; j <= i ; j++) {
        :
    }
}
```

i	j	#iterations
1	[1 1]	1
2	[1 2]	2
:	:	:
N	[1 N]	N
		$\frac{N*(N+1)}{2}$

TC = $O(N^2)$

9)
```
for (i = 1 ; i <= N ; i++) {
    for (j = 1 ; j <= 2^i ; j++) {
        :
    }
}
```

i	j	#iterations
1	[1 2^1]	2^1
2	[1 2^2]	2^2
:	:	:
N	[1 2^N]	2^N
		$2^1 + 2^2 + 2^3 + \ldots + 2^N$

$a = 2 \quad r = 2 \qquad = \frac{2*(2^N - 1)}{(2-1)} = 2*(2^N - 1)$

$TC = O(2^N)$

$$S = \frac{a*(r^N - 1)}{(r-1)}$$

Compare #iterations

	Faisal	Mouriks
	Algo 1	Algo 2
#iterations →	$100 * \log(N)$	$N/10$
$N <= 3500$		✓
$N > 3500$	✓	✓

Real World → 1) Live match views ≈ 10 M

2) Highest watched youtube video ⇒ "Baby Shark" ≈ 13 B

↓

very large data

Asymptotic Analysis / Big O

Analysing performance of algorithms for large inputs.

Calculation of Big O → Rate of growth of function

✓ 1) calculate #iterations wrt input size

→ 2) Ignore lower order terms

→ 3) Ignore constant coefficient

Eg → #iterations → $(4N^2 + 3N - 1)$ → $O(N^2)$

$4N^2$

N^2

→ $\log(N) < \sqrt{N} < N < N * \log(N) < N\sqrt{N} < N^2 < N^3 < 2^N < N! < N^N$

$2N^3 - 4\log(N) + 1500 * N\sqrt{N}$ → $O(N^3)$

Time complexity

Rate of growth of ~~time~~ #iterations wrt input.

TC = O(#iterations)

$O(4N + 3N\log(N) + 1)$ → $O(N\log(N))$

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

Why to neglect lower order terms?

N	#iterations $(N^2 + 10 * N)$	% contribution of lower order term
10	$10^2 + 10 * 10$ $= 100 + 100 = 200$	$\frac{100}{200} * 100 = 50\%$
100	$100^2 + 10 * 100$ $= 10000 + 1000$ $= 11000$	$\frac{1000}{11000} * 100 \approx 9\%$

10000	$10000^2 + 10 * 10000$ $= 10^8 + 10^5$	$\frac{10^5}{(10^8 + 10^5)} * 100 \simeq \frac{10^7}{10^8} = 0.1\%$

% contribution of lower order term is significantly low for large inputs.

Issues with Big O

1)

	Algo 1	Algo 2
# iterations →	$5000 * N$	N^2
TC →	$O(N)$ ✓	$O(N^2)$
$N = 1000$ →	$5 * 10^6$	10^6 ✓

Big O is only useful to understand algo wrt very large inputs.

---

2)

	Algo 1	Algo 2
# iterations →	$5000 * N$	$2 * N$ ✓
TC →	$O(N)$	$O(N)$

For algorithms where $O(-)$ is same, we have to check # iterations to decide which algo. is better.

TLE (Time Limit Exceeded)

Amazon

Aseer → 2 questions in 1 hour

understand the question
find bruteforce
use examples to find observations
find optimised code
code it (TLE)

Expected TC → Helps

Why TLE occurs

online coding platform → 1 GHz processing speed

Expected time = 1 sec

10^9 instructions per sec

a = 2

b + c

!

Let say 1 iteration takes 10 to 100 instructions.

10 instructions → 1 iteration

⇒ 10^9 instructions → 10^8 iterations

100 instructions → 1 iterations

10^9 instructions → 10^7 iterations

If time limit is 1 sec

then # iterations → 10^7 to 10^8

TC = $O(N^2)$

$N = 10^3$	→ 10^6	✓
$N = 10^5$	→ 10^{10}	→ TLE
$N = 10^4$	→ 10^8	→ may or may not give TLE.

Constraints

$1 <= N <= 10^5$ largest TC ≈ O(N log N) / O(N)...

largest $N = 10^5$
