$1000\, n\log n = O\left(\frac{n\log n}{1000}\right)$ (T/F)

$1000\, n\log n \leq c\, \frac{n(\log n)}{1000}$ ✓

1000

'n' → input size.

Whenever leading terms of functions differ only by constant.

The O, Ω, Θ can be applied

Ex: $\frac{3n^2+2n+1}{3n^2+2n+1} = O(n^2)$
$3n^2+2n+1 = \Omega(n^2)$
$3n^2+2n+1 = \Theta(n^2)$

} not mathematically these are asymptotic

If the running time of an algo is $4n^2 + n + 1$.

then $= O(n^2)$ — Tight
$= \Omega(n^2)$
$= \Theta(n^2)$
$= O(n^3)$ — loose
$= \Omega(n)$

Tight upper bound means we cannot go below that
Tight lower bound means we can't go above that.

$4n^2 + n + 1 = \Omega(n)$

$4n^2 + n + 1 \geq n$

$\sqrt{\log n}$, $\log\log n$

apply $\log$.

$\frac{1}{2}\log\log n \Rightarrow \log\log\log n$

$\sqrt{\log n} > \log\log(n)$

$\sqrt{\log n} = \underline{\underline{\omega}}(\log\log n)$

$= \underline{\underline{\Omega}}(\log\log n)$

2^{n+1} 2^n

$2^n \cdot 2$ 2^n

differ by a constant

$2^{n+1} = O(2^n)$

$= \Omega(2^n)$

$= \Theta(2^n)$

2^{2n} 2^n

$2^n \cdot 2^n$ 2^1

$2^n > 1$

$2^{2n} > 2^n$

$2^{2n} = \omega(2^n)$

$2^{2n} = \Omega(2^n)$

$=$

$2^{2n} > 2^n$

$2^{2n} = \omega(2^n)$

$2^{2n} = \Omega(2^n)$

$1/n < 1$

$1/n \leq o(1)$

$1/n = O(1)$ ✓

↓

decreasing function

as $n \to \infty$, $1/n \to 0$

100000, 1

all constants are asymptotically same.

$100000 = O(1)$ ✓

$100000 \leq c \cdot 1$

$100000 = \Omega(1)$ ✓

$100000 = \Theta(1)$ ✓

$n^{\log n} < (\log n)^n$ ✓

apply log ✓

$\log n \times \log n = (\log n)^2$

~~n log~~ $n^{\log n} < (\log n)^n$

$n^{\log n} = O(\log n)^n$

$n^{\log n} = O(\log n)^n$

$n \log n \log n$.

$n = 2^{1024} = 2^{2^{10}}$

$(\log_2 2^{2^{10}})^2$

$(2^{10})^2$

2^{20}

$n \log_2 \log_2 2^{2^{10}}$

$= n \log_2 2^{10}$

$= 2^{1024} * 10$

$n^c < c^n$

c – Constant

polynomial

Exponential.

apply log.

~~$c \log n$~~ ~~$n \log c$~~

$\log n < n$

$n^c < c^n$

$n^c = o(c^n)$

$= O(c^n)$

$f(n) = O(f(n))$ T/F T

$f(n) = \Omega(f(n))$ T/F T

$f(n) = \Theta(f(n))$ T.

$f(n) \stackrel{?}{=} O(f(n/2))$

Is $f(n)$ always equal to $O(f(n/2))$

Case(i) Let $f(n) = n^2 + 2$

$f(n/2) = \frac{n^2}{4} + 2$

$f(n) = O(f(n/2))$

$n^2 + 2 = O\left(\frac{n^2}{4} + 2\right)$

$n^2 + 2 \leq C \cdot \left(\frac{n^2}{4} + 2\right)$

C ↓ 10000000

In this Case $f(n) = O(f(n/2))$

Case 2: Let $f(n) = 2^{2n}$

$f(n/2) = 2^{2(n/2)} = 2^n$.

$2^{2n} = O(2^n)$

$2^n \; 2^n$

2^n

2^n

$\vdots$

In this case $f(n) \neq O(f(n/2))$

☺

$2^{2n} = O(2^n)$

$2^{2n} \leq c \cdot 2^n$

$2^n \leq 1000000002^n$

but when $n > 1000000000000000$

$(n+3)^{25} = O(n^{25})$

$(n+3)^{25} \leq c \cdot n^{25}$

$\frac{(n+3)^{25}}{n^{25}} \leq c$

$\left(\frac{n+3}{n}\right)^{25} \leq c$

$\left(1+\frac{3}{n}\right)^{25} \leq c$

$n \to \infty$

$(1)^{25} \leq c$

1 ∩ 2 ∩ 3 ⋯

$(n+3)^{25} = \Omega(n^{25})$

$(n+3)^{25} = \Theta(n^{25})$

$O, \Omega \Rightarrow \Theta.$

$f(n) = O((f(n))^2)$ T ∧ false.

Case(i): $f(n) = n \rightarrow$ increasing ∴ it is not always true

$(f(n))^2 = n^2$

$f(n) = O(f(n)^2)$

$f(n) = O((f(n))^2)$ not always true..

$f(n) = 1/n \rightarrow$ decreasing.

$f(n) \neq O((f(n))^2)$

$n \uparrow \quad f(n) = 1/n$ dec ↓

In practice $T(n) = \frac{1}{n}$

as input increases, time taken decreases.

It is not practical.

It is just for theory

decreasing functions are just for theory.

$f(n) = O((f(n))^2)$ where $f(n) > 1$

$f(n)$ is increasing function

$\therefore \quad f(n) = O((f(n))^2)$

$f(n) = O\left((f(n))^2\right)$ where $f(n) < 1$

$f(n)$ is decreasing function.

$\therefore \quad f(n) \neq O\left((f(n))^2\right)$

→ If $f(n) = O(g(n))$ then $2^{f(n)} = O(2^{g(n)})$

Is it always true.

It is not always true

False

Case: $f(n) = 2n$ and $g(n) = n$

$f(n) = O(g(n))$

$2^{f(n)}$ $2^{g(n)}$

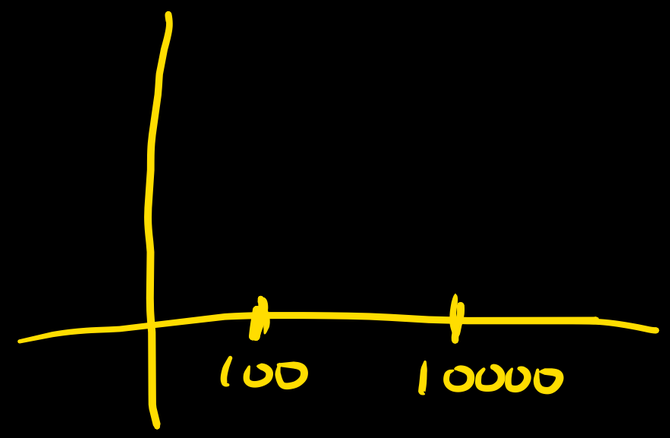
2^{2n} 2^n

$2^n \cdot 2^n$ 2^n

$2^{f(n)} \neq O(2^{g(n)})$

$$f(n) = \begin{cases} n^3 & \text{if } 0 < n < 100 \\ n^5 & \text{if } n \geq 100 \end{cases}$$

$$g(n) = \begin{cases} n^9 & \text{if } 0 < n < 10000 \\ n^3 & \text{if } n \geq 10000 \end{cases}$$



When all the functions stabilises, $n \geq 10000$

$f(n) = n^5 \quad \& \quad g(n) = n^3$

$f(n) = \Omega\, g(n)$

$= \omega\, g(n)$

How can we answers such questions in
exam. of job interview.

Ans: I am discussing everything needed.

Properties of asymptotic notations:

1) $f(n) = O(f(n))$ reflexive.

$O, \Omega, \Theta, o, \omega$

2) Symmetric property

if $f(n) = \Theta(g(n))$ then $g(n) = \Theta(f(n))$

$\Theta, \delta, \Xi, \delta, \Xi$

3) if $f(n) = O(g(n))$ and $g(n) = O(h(n))$

then $f(n) = O(h(n))$ [transitive property]

$O, \Omega, \Theta, o, \omega$

4) $f(n) = O(g(n))$ and $h(n)$ is a +ve function

$$f(n) \cdot h(n) = O(g(n) \cdot h(n))$$

$O, \Omega, \Theta, o, \omega$

5) If 1) $f(n) = O(g(n))$

2) $d(n) = O(e(n))$

$$f(n) + d(n) = O(g(n) + e(n))$$

$$f(n) \cdot d(n) = O(g(n) \cdot e(n))$$

ಅಂದರೆ

Algorithms / programs

Iterative

Recursive

analysing iterative and recursive programs are different.

Analysing iterative programs :-

1)
```
main ()
{
    a = b+c;
}
```

} pseudo code

3 var.

TC = O(1)

SC = O(1)

Constant

2)

```
main()
{
    for (int i=0; i<n; i++)
    {
        num += 1;
    }
}
```

→ O(n)

$T(C) = O(n)$

$SC = O(1)$

for (i=0 ; i<n ; i=i+2) { num = num +1 }

Let us assume the loop runs for k times. We want to find k.

iteration	1	2	3	4 ..... K	k+1 (loop fails)
value i	0	2	4	6 ... 2(k-1)	2k
		2(2-1)	2(3-1)	2(4-1)	

∴ loop runs for n/2 times
O(n) ✓
Ω(n) ✓
Θ(n) ✓

Why loop fails $i = n$ → ①.

the value of i when loop failed = 2k

$i = 2k$ in ① $\quad 2k = n \Rightarrow k = n/2$ ✓

```
for (i=1 ; i<n; i *= 2) { num += 1 };
```

Let the loop run k times. we want find k

iteration	1	2	3	4	....	k fails
i	1	2	2^2	2^3		2^{k-1}

i: 1 → (n-1) (n) ✓

$i = n$

$2^{k-1} = n$

$k-1 = \log_2 n$

$k = O(\log_2 n)$ ✓

$\Omega(\log_2 n)$

$\Theta(\log_2 n)$

for(int i=n; i>2; i=√i) {Constant time}

Let loop runs k times. we have to find 'k'

iteration	1	2	3	4	.....	k	k+1 (fails)
i	n	$n^{1/2}$	$n^{1/2^2}$	$n^{1/2^3}$		$n^{1/2(k-1)}$	$n^{1/2k}$

$O(\log\log n)$ ✓

why loop fails $i=2 \Rightarrow (1)$

Value of i when loop fails $i = 2$ $n^{1/2^k}$

$\therefore \quad n^{1/2^k} = 2$

$\frac{1}{2^k} \log n = \log_2 2 \Rightarrow 2^k = \log n$

$\Rightarrow k = \log\log n$

```
int i = n
while (i > n-1) {
    numb += 1
    i = i/2
}
```

i $n \to 1$

$O(\log_2 n)$

iteration	1	2	3	...	k	k+1 times
i	n	$n/2$	$n/2^2$		$n/2^{k-1}$	$n/2^k$

loop runs $i = 1$

$$\frac{n}{2^k} = 1 \Rightarrow n = 2^k$$

$$k = \log_2 n$$

```
fol ( i= 1 ; i^2<n  i++)
{   a=b+c;
}
```

$i^2 < n$

$i < \sqrt{n}$

$i = 1$ to $\sqrt{n}$.

$O(\sqrt{n})$

Space Complexity is total memory it took

i, a, b, c, n

$5 \times 4 = 20 = O(1)$

```
for ( i = n/2 ; i < n ; i = 2 * i )
  {  a = b + c
  }
```

iteration 1 (2) halts.

i n/2 n

O(1)

↓

Constant time.

```
for (i=0 ; i<n ; i++)
  {
     for (j=0 ; j<m ; j++)
       {
          numb+1=1
       }
  }
```

running loop.

$n(m-1)$

$i=0$	$i=1$	----	$i=n$
$j=0\ 1\ 2 \cdots m-1$	$j=0,1,2 \cdots m-1$	----	$j=0,1,2 \cdots m-1$

```
i = 1
while (i<n) {
    int j=1;
    while (j<n) {
        num += 1;
        j = j*2;
    }
    i = i*2
}
```

$\log_2 n$

$\log_2 n$

$(\log_2 n)^2$

```
fn(int i=n^2; i>1; i=i/2)
{
    num+1=1
}
```

$i>1 \Rightarrow$ $i=1$ → loop stops

iteration	1	2	3	....	K	(K+1) times
i	n^2	$n^2/2$	$n^2/4$		$\frac{n^2}{2^{K-1}}$	$\frac{n^2}{2^K}$

$i=1$

$\frac{n^2}{2^K} = 1 \Rightarrow n^2 = 2^K$

$\Rightarrow K = \log n^2$

$K = 2\log n$

$TC = O(\log n)$

$SC = O(1)$