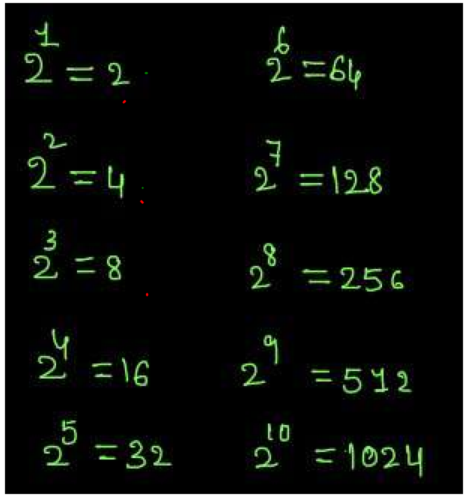
Welcome

To

Master Class on T.C && S.C

-- Venu



$2^1 = 2$	$2^6 = 64$
$2^2 = 4$	$2^7 = 128$
$2^3 = 8$	$2^8 = 256$
$2^4 = 16$	$2^9 = 512$
$2^5 = 32$	$2^{10} = 1024$

$1\,\text{kg} = 1000\,\text{g} = \underset{k}{10^3}\,\text{g}$

$1024\,B = 1\,KB \cong 10^3\,B$

↳ $1024 = 2^{10}$

→ $1\,KB \cong 1000\,B \cong 10^3\,B$.

→ $1\,MB \cong 1000\,KB$

$\cong 1000 \times 1000\,B$

$\cong 10^6$ Bytes.

$\therefore 1\,MB \cong 10^6\,B$.

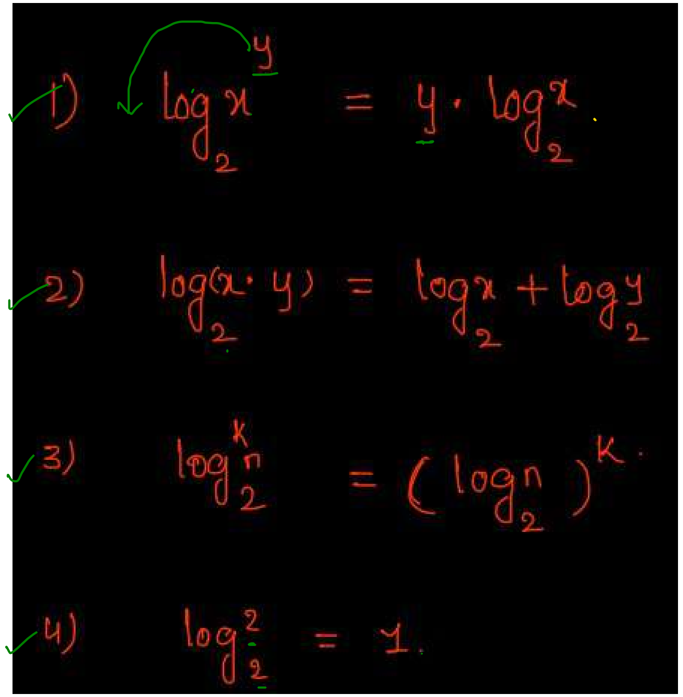
$1 \leq N \leq 10^6$ ✓

* Units of Computer Memory Measurements

1 Bit ✓	= Binary Digit
8 Bits	= 1 Byte ✓
1024 Bytes	= 1 KB [Kilo Byte]
1024 KB	= 1 MB [Mega Byte]
1024 MB ✓	= 1 GB [Giga Byte]
1024 GB	= 1 TB [Terra Byte]
1024 TB	= 1 PB [Peta Byte]
1024 PB	= 1 EB [Exa Byte]
1024 EB	= 1 ZB [Zetta Byte]
1024 ZB	= 1 YB [Yotta Byte]
1024 YB	= 1 Bronto Byte
1024 Brontobyte	= 1 Geop Byte

Geop Byte is the Highest Memory.

Some of the basic Math formulas



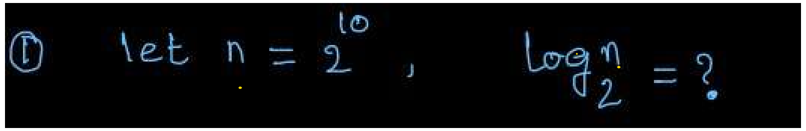
✓1) $\log_2 x^y = y \cdot \log_2 x$.

✓2) $\log_2 (x \cdot y) = \log_2 x + \log_2 y$

✓3) $\log_2^k n = (\log_2 n)^k$.

✓4) $\log_2 2 = 1$.

✓5) $n! = n \times n-1 \times n-2 \cdots \times 1 \cong n^n$



① let $n = 2^{10}$, $\log_2 n = ?$

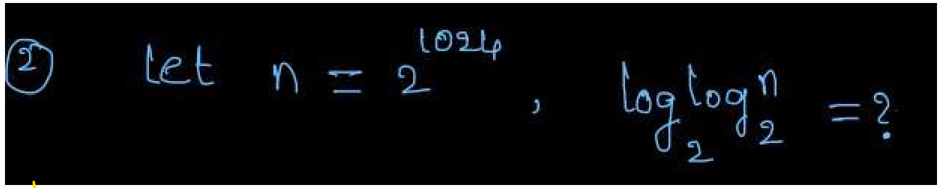
Algo :- $\log_2$ ✓

↑

Base

Sol)

$$\log_2 2^{10} = 10 \cdot \log_2 2 = 10 \times 1 = 10 ✓$$



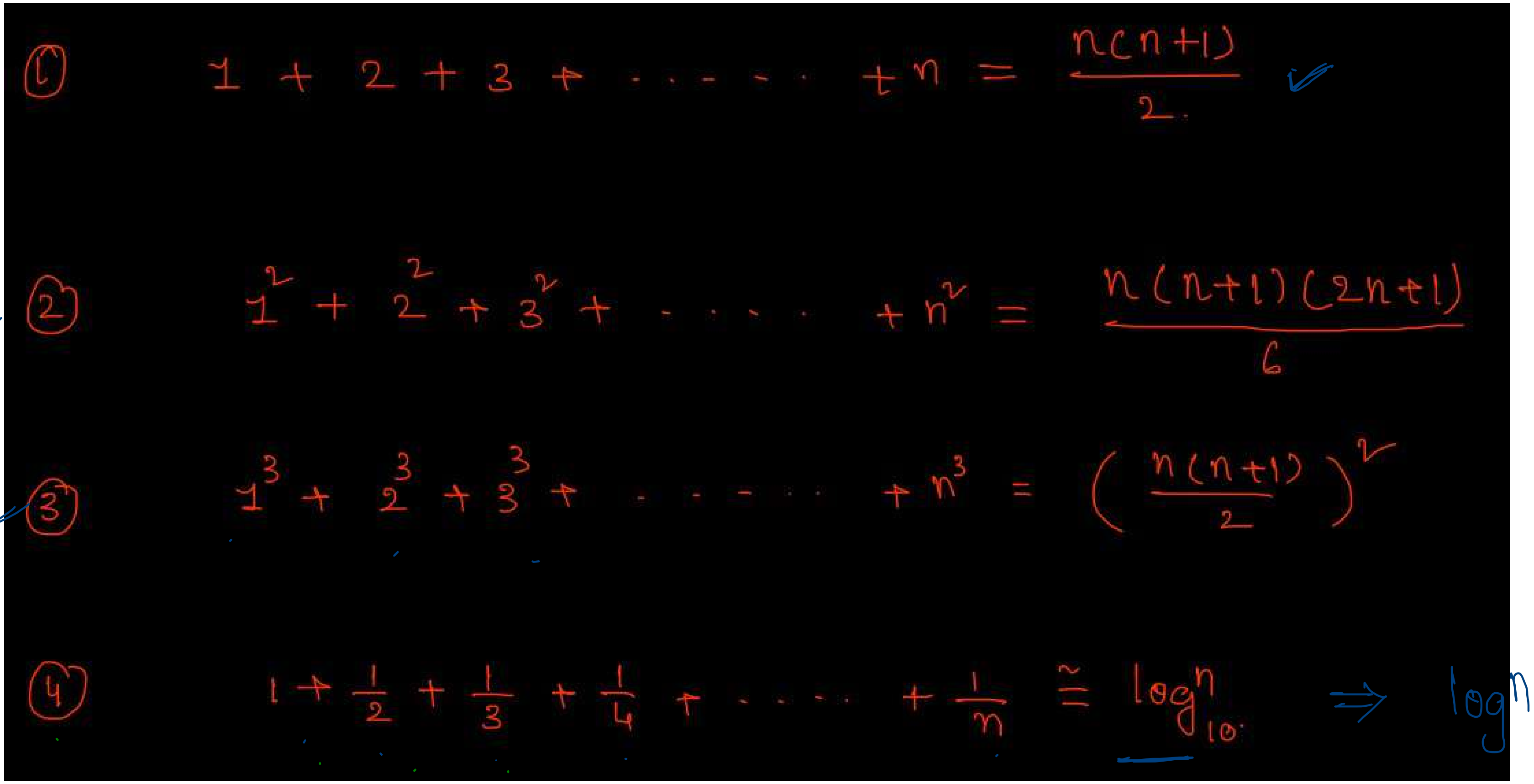
② let $n = 2^{1024}$, $\log_2 \log_2 n = ?$

Sol

$$\log_2 . \underbrace{\log_2 2^{1024}}_{1024} = \log_2 1024 = \log_2 2^{10}$$

$$= 10 \cdot \underbrace{\log_2 2}_{1}$$

$$= 10 ✓$$



✓ ① $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$ ✓

✓ ② $1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$

✓ ③ $1^3 + 2^3 + 3^3 + \cdots + n^3 = \left(\frac{n(n+1)}{2}\right)^2$

* ④ $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} \simeq \underline{\log_{10}^{n}}$ ⇒ $\log^n$ series.

* ⑤. ✓ $\underline{(n)\times(n-1)\times(n-2)\times\cdots 1} = n! \overset{\uparrow}{\simeq} n^n$

$n! < n^n$ ✓

$$1+2+3+\cdots+n = \frac{n(n+1)}{2}$$

① $1 + 2 + 3 + \cdots + n-1 = ?$

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

② $\log 1 + \log 2 + \log 3 + \log 4 + \cdots \log n = ?$

$$\log(a\cdot b) = \log a + \log b$$

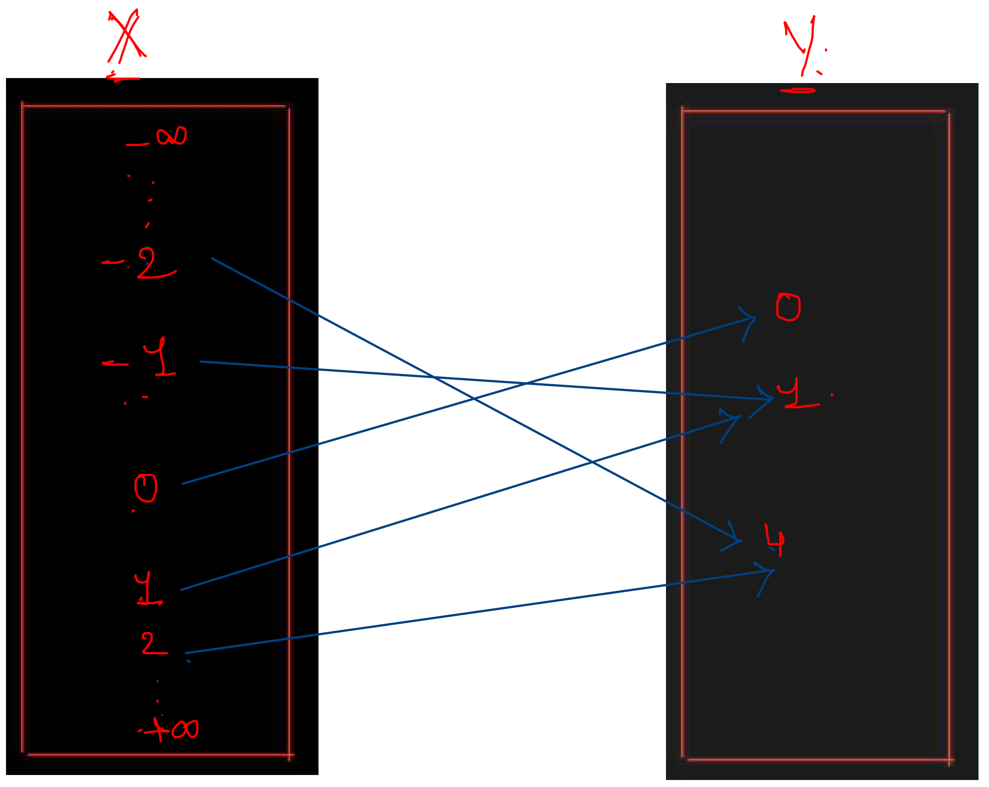
$$= \log(1 \times 2 \times 3 \times \cdots \times n)$$

$$= \log(n!)$$ ✓

Function :-

$f(x) = x^2$

a function is an expression, defined as : from set X to set Y, assigns each element of X to the exactly one element of Y



$\boxed{f(x) = x^2}$ $f(n) = n^2$

$f(-2) = (-2)^2 = 4$

$f(-1) = (-1)^2 = 1$

$f(0) = 0$

$f(-2) = 4$

$f(2) = 4$

1. Algorithms T.C is very much related to functions in math
2. The following functions are commonly used in Algorithms

Sno	Function Name	Function Expression
1	Constant	1
2	Logarithmic	log(n)
3	Square root	√n
4	Linear	n
5	Linearithmic	n.log(n)
6	Quadratic	n^2
7	Cubic	n^3
8	Exponential	2^n
9	Factorial	n!

$+O(\quad)$

→ $f(n) = \log(n)$

$f(n) = n^3$

functions

for One problem Many Solution's are possible

Solution-1 ✓

Solution-2 ✓

Solution-3 ✓

....

Solution-n

Ramesh → S_1

Suresh → S_2

How to choose which solution is best? When we have more than one solution

$f_1 = \underline{2^n}$ $\quad$ $f_2 = n^2$ $\quad$ $\Rightarrow$ ~~$O()$~~

$\underline{R}$ $\quad$ $\underline{S}$

⇒ 1.Cancel all the common terms in the functions which you are comparing

⇒ 2.Apply log to all the function which you are comparing [if requires]

3.Put very large values in it [let n=2^1024, 2^2^1024 etc..] $\quad 2^n \Rightarrow \log_2$

$\rightarrow$ less time, Better $(\underline{n>0})$

① $^* f(n) = n^2 \quad (R)$

$g(n) = n^3 \quad (S)$

$n^2 \qquad n^3$

$\cancel{n^2} \qquad \cancel{n^2} \times n$

$\underline{1} \qquad \underline{n}$

let $n = 2^{1024}$

$\boxed{\underline{1}} \qquad 2^{1024}$

$f(n) < g(n)$

② $f(n) = 2^n \cdot (R)$

$g(n) = n^2 \cdot (S)$

$\underline{2^n} \qquad \underline{n^2}$

$\log_2 2^n \qquad \log_2 n^2$

$n \cdot \underbrace{\log_2 2}_{1} \qquad 2 \cdot \log_2 n$

$n \qquad 2 \cdot \log_2 n$

let $n = 2^{1024}$

$\downarrow$

$\underline{2^{1024}} \qquad 2 * \log_2 2^{1024}$

$\underline{(1024 \times 2)} \times 1$

$g(n) < f(n)$

③ $f(n) = 2^n$

$g(n) = 3^n$

$2^n \qquad 3^n$

let $n = 100$

$\underline{2^{100}} \qquad \underline{3^{100}}$

$\underline{f(n)} < g(n)$ ✓

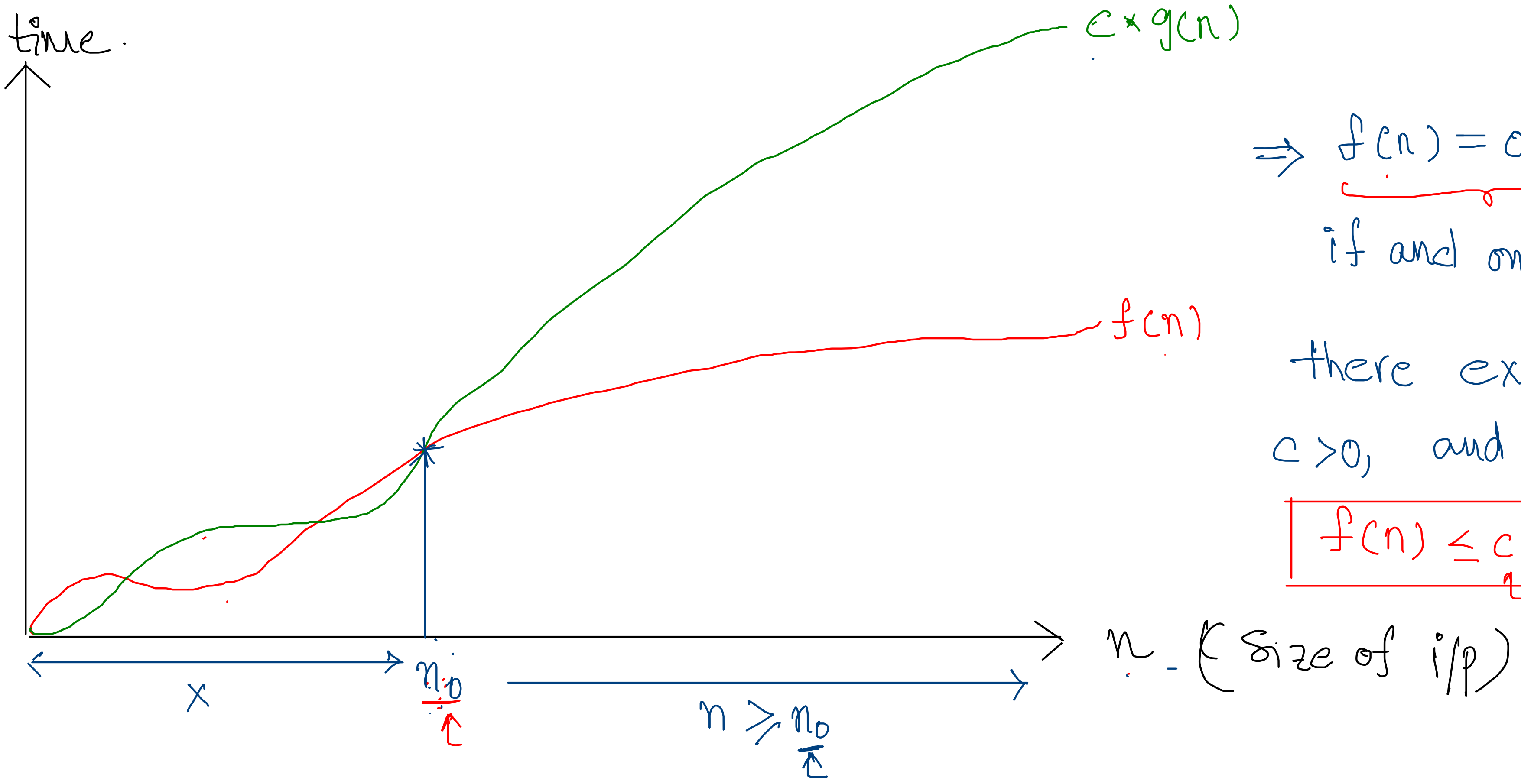
④ $f(n) = n.$

$g(n) = (\log_2^n)^{100}$

⑤ $f(n) = \underline{n}^{\sqrt{n}}$ $g(n) = \underline{n}^{\log_2^n}$

Big-Oh [O( )] : Order of :-

[Math f(n) + O( )] T.C



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

if and only if

there exists some

$c > 0$, and $n \geq n_0$

$$\boxed{f(n) \leq c \cdot g(n)}$$

let

$$f(n) = 2\underset{\underset{*}{\uparrow}}{n^2} + n + 3 \quad ; \quad f(n) \overset{?}{=} O(\underline{n^2})$$

$\underline{c > 0}$

$c = 1 \times$

$=$

$\vdots$

$\underline{\text{sol}}$

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

$$f(n) \le c \cdot g(n), \quad \underline{c} > 0$$

$$n \ge \underline{n_0}$$

$$2n^2 + n + 3 \le c \cdot n^2$$

let $c = 10$

$$(2n^2 + n + 3) \le 10 \cdot n^2$$

$n = 1 \Rightarrow 2 + 1 + 3 \le 10$ ✓

$n = 2 \Rightarrow 8 + 2 + 3 \le 40$ ✓

$\vdots$

$\boxed{n_0 = 1}$, $\boxed{c = 10}$ $\therefore f(n) = O(n^2)$

Ex:-

$$\left.\begin{aligned} f(n) &= n^2 \\ g(n) &= 2^n \end{aligned}\right\} \quad f(n) \overset{?}{=} O(g(n)) \qquad \Rightarrow \quad g(n) = O(f(n)) \quad X$$

sol)

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

$f(n) \le c \cdot g(n), \quad c > 0,$

$n \ge n_0$

$n^2 \le c \cdot 2^n$

$n^2 \le 5 \cdot 2^n$ let $c = 5$ ✓

$n = 1 \Rightarrow 1 \le 5 * 2$ ✓

$n = 2 \Rightarrow 4 \le 5 * 4$ ✓

$n = 3 \Rightarrow 9 \le 40$ ✓

$n_0 = 1$ ✓

$\therefore f(n) = O(g(n))$ ✓

✓ Consider the following two functions:

$$g_1(n) = \begin{cases} n^3 & \text{for } 0 \le n \le 10,000 \\ n^2 & \text{for } n > 10,000 \end{cases}$$

$$g_2(n) = \begin{cases} n & \text{for } 0 \le n \le 100 \\ n^3 & \text{for } n > 100 \end{cases}$$

Which of the following is true?

✓ A. $g_1(n)$ is $O(g_2(n))$

✓ B. $g_1(n)$ is $O(n^3)$

✗ C. $g_2(n)$ is $O(g_1(n))$

✗ D. $g_2(n)$ is $O(n)$ $\Rightarrow$ $g_2(n) \le c \cdot n$

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

$$\Rightarrow \quad f(n) \le c \cdot g(n)$$

0, 100, 10,000, >10,000

*

	0 - 100	100 - 10,000	>10,000
$g_1(n)$	n^3	n^3	n^2
$g_2(n)$	n	n^3	n^3

finally $\quad g_1(n) = O(g_2(n))$

$$g_2 > g_1$$

$$g_1(n) = O(g_2(n))$$

(B) :- $g_1(n) \le c \cdot n^3$

11:00 AM

When nested loop

↳ dependency (?)

```
O.L → for(i=1; i<=n;i++)  → i : loop-variable.
      {
I.L     → for(j=1;j<=n/2;j++)
          {

                c=c+1

          }
      }
```

```
n   → for(i=1; i<=n;i++)
      {
n/2     → for(j=1;j<=n/2;j++)
          {

                c=c+1 => O(1)

          }
      }
```

No-need of exact value,

Approximation enough.

$$n * \frac{n}{2} * 1 = \frac{n^2}{2} = \cancel{\frac{1}{2}} \cdot n^2$$

$$\boxed{O(n^2)}$$

```
for(i=1;i<=n;i++)  → n
{
        for(j=1;j<=n/4;j++)  → n/4
        {
                for(k=1;k<=n;k++)  → n
                {

                        print("*")  → O(1)
                }
        }
}
```

$$\Rightarrow n * \frac{n}{4} * n * 1$$

$$\Rightarrow \cancel{\frac{1}{4}} * n^3 \Rightarrow \underline{O(n^3)}$$

```
for(i=1;i<=n;i++)  → n
{
        for(j=1;j<=n/4;j++)  → n/4
        {
                for(k=1;k<=n;k++)  → 1
                {
                        print("*")
                        break; ✓
                }
        }
}
```

$$\Rightarrow n * \frac{n}{4} * 1$$

$$\Rightarrow \cancelto{1}{\frac{1}{4}} \cdot n^2 \Rightarrow \underline{O(n^2)}$$

```
① for(i=1; i<=n; ++i)  → n
  {
  ②    for(j=1; j<=n; j++)  → n
        {
        ③    for(k=n/2; k<=n; k=k+n/2)  → 2 times
              {

                    ~~c=c+1~~
                    P(*)
              }
        }
  }
```

$\Rightarrow \quad n * n * 2$

$\cancel{2} n^2 \Rightarrow O(n^2)$

③ $\underline{k}$

$\cancel{\frac{n}{2}} \quad \frac{n}{2} + \frac{n}{2} = \cancel{n} \quad \frac{3n}{2}$

$\times$

$\underline{*\ *}$

⇒ Approximate ✓

Some $n = \underline{16}$

✓ GP

```
⇒ i=1
  while(i<=n)
  {       → P(*)
          i=i*2
  }
```

⇓

$i = i * 3$

↳ $O(\log_3^n)$

→ $i = i * K$ ⇒ $(\log_K^n)$

$i = \cancel{1} . \cancel{2}\ \cancel{4}\ \cancel{8}\ \cancel{16}\ 32$

*

* * * * *

$1 \downarrow \times 2$

$2 \downarrow \times 2$ ⇒ $\underline{O(\log_2^n)}$

$4 \downarrow$

$8 \downarrow$

16

$16 \downarrow /2$

$8 \downarrow /2$

$4 \downarrow$

$2 \downarrow$

1

⇒ $O(\log_2^n)$

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

$\longrightarrow n$

$\Rightarrow \underbrace{\cancel{1}\quad \cancel{2}\quad \cancel{4}\quad \cancel{8}\quad 16 \ldots\ldots}_{\log_2^n}$

$O(n * \log_2^n)$

```
let m=2^n

for(i=1; i<=n; i++)
{
        for(j=1; j<=m; j=2*j)
        {
                c=c+1
        }
}
```

✓

```
for(i=1; i<=n; i++)   → n
{
        for(j=1; j<=n; j++)   → n
        {
                c=c+1
        }
}
```

$O(n^2)$

⇒ dependency

```
for(i=1; i<=n; i++)
{
        for(j=1; j<=i; j++)
        {
                c=c+1 // P(*)
        }
}
```

n=5

⇒ i=6 ✗

i=1

j = ~~1~~ 2 :

*

i=2

j = ~~1~~ ~~2~~ 3

* *

i=3

j = ~~1~~ ~~2~~ ~~3~~ 4

* * *

i=4

* * * *

i=5

5(*)

when n=5

$$1 + 2 + 3 + 4 + 5 \Rightarrow \frac{n(n+1)}{2} \Rightarrow \frac{n^2+n}{2}$$

$O(n^2)$

```
function fun(n,m)
{
        for(i=1;i<=n;i++)
        {
                for(j=i+1; j<=m; j++)
                {
                        print("*")
                }
        }
}
```

→ i

let n=5, m=4

i=1: j = ~~2~~ ~~3~~ ~~4~~ 5 — * * *

i=2: j = ~~3~~ ~~4~~ 5 — * *

i=3: j = ~~4~~ 4 5 — *

i=4: j = 5 ×

i=5: j = 6 ×

m=4   3 + 2 + 1

$\frac{m(m-1)}{2} \Rightarrow m^2$ $[\ O(n*m)\ ]$

```
for(i=1; i<=n; i++)
{
        for(j=1; j<=n; j=j+i)
        {
                c=c+1
        }
}
```

P(*)

$n=10$

$i=1$

$j=1\ 2\ 3 \ldots\ldots 10$

$* \ * \ * \ * \ * \ldots$

$10(*)$

$\left\lceil \frac{10}{1} \right\rceil$

$i=2$

$j=1\ 3\ 5\ 7\ 9\ 11$

$* \ * \ * \ * \ *$

$5(*)$

$\left\lceil \frac{10}{2} \right\rceil$

$i=3$

$\left\lceil \frac{10}{3} \right\rceil \ldots\ldots$

$$\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \ldots\ldots + 1$$

$$n\left[\underbrace{1 + 1/2 + 1/3 + \ldots + 1/n}_{\log_{10} n}\right] \Rightarrow O(n*\log n)$$

```
i=n
while(i>=0)
{
    j=1
    while(j<=n)
    {
            j=j*2
    }
    i=i/2
}
```

$\Rightarrow \log_2^n$

$\underline{\quad} \; n$

$\log_2^n$

$n * \log_2^n$

$$\underset{O.L}{\underline{\log_2^n}} * \underset{I.L}{\underline{\log_2^n}} = (\log_2^n)^2$$

```
for(i=1; i<=n; i++)
{
        j=1

        while(j<=n)
        {
                j=2*j
        }

        for(k=1;k<=n;k++)
        {
                c=c+1
        }
}
```

$\Rightarrow n$

$\log_2^n \Rightarrow ①$

$\} \; n \Rightarrow ②$

$n(\; ① + ② \;)$

$n(\; \log_2^n + n \;)$

$= \underline{n \cdot \log_2^n} + \underline{n^2}$

$\Rightarrow \underline{O(n^2)}$ ✓

$\underline{n^2}$ $\quad$ $\underline{n\log_2^n}$

$\cancel{n} * \underline{n}$ $\quad$ $\cancel{n}\log_2^n$

$\underline{n}$ ✓ $\quad$ $\underline{\log_2^n}$

```
function fun(n)
{        q = 0
        for(i=1;i<=n;i++)
        {
            p=0
            for(j=n; j>1; j=j/2)
            {
                ++p
            }

            for(k=1; k<p; k=k*2 )
            {
                ++q
            }
        }
}
```

⇒ GATE

for(i=1;i<=n;i++) ⇒ n

for(j=n; j>1; j=j/2) ⇒ $\log_2^n$ ⇒ $p = \log_2^n$

for(k=1; k<p; k=k*2) ⇒ $\log_2^p$ ⇒

→ $\log_2^n$

$+$

$\log\log_2^n$

$n(\log_2^n + \log\log_2^n) \Rightarrow \underset{①}{n\cdot\log_2^n} +$

$\underset{②}{n\cdot\log\cdot\log_2^n}$

∴ T.C : $O(n\cdot\log_2^n)$

✓ Assume arr.sort() will take T.C as nlog(n)

Algo/sol : in-built ✓

```
function fun(arr,n)
{
   1. arr.sort()          ⇒ n·log₂n

   2. for(i=1;i<=n;i++)   ⇒ n
      {
           console.log(arr[i])
      }
}
```

$$n + n \cdot \log_2^n \Rightarrow O(n \cdot \log_2^n)$$

Let T: be the number of test cases (some $t>0$)

```
while(T>0)
{
        for(i=1;i<=n;i++)
        {
                1. arr.sort()
                2. j=1
                while(j<=n)
                {
                        j=j*2
                }
        }
        T--
}
```

for(i=1;i<=n;i++) → n

arr.sort() → $n \cdot \log_2 n$

$\log n$

T

n

$$T * (n * (n \cdot \log_2 n + \cancel{\log_2 n}))$$

$$\boxed{O(T \cdot n^2 \cdot \log_2 n)}$$

✓ Space Complexity [S.C]

program

$S.C(p) = C + I.C$

C : Constant space [all variables, data structures with fixed size] ⇒ O(1)

I.C : Instance Characteristics

-> I.C includes space for all such variables and data structures whose size is not known before

* NOTE:-

- > The space requirement of any Algorithm / Program is bifurcated into two parts

✓ 1. Space for inputs

✓ 2. work space requirements

w.s component is the space used during " computation " of the Algorithm/ Porgram

i.e , any space that you used, other than storing inputs

```
function fun(a,b,c)
{
        var p=a
        var q=b
        var r=c
        console.log(p+q+r)
        return a+b+c
}
```

p

q

r

⇒ O(1)

Write a program to find the sum of all the elements to left of every element in array

```
function sumOfLeft(arr,n)
{
   left=[] ==>
      let left=new Array(n)

      // code you take care


}
```

size

x → ?

JS → O(n)

~~O(n)~~

i/p

arr

0	1	2	3	4	5
10	20	30	40	50	60

n=6

o/p

left

0	1	2	3	4	5
0	10	30	60	100	150

n=6

right

S.C :- ?

C + I.C

O(1) + O(n) ⇒ O(n)

n+n = 2n

O(n)

```
function fun(n)
{
        if(n==1)
                return 1
        else
                return n*fun(n-1)
}

main()
{
  temp=fun(5)
  print(temp )
}
```

S.C:

⇒ Recursion

⇓

stack-space ⇒ ✓

✓Note :-

In general work space requirements for Algorithm is the order of time complexity

↳ other than i/p storing,

what is the extra space you are using

✳ ✳ $\boxed{W.S(Algorithm) = O(T.C)}$

T.C

Ex :- Prob :- $\underbrace{O(n^2)}$ ✓

$S.C \leq c.n^2$

↑

if

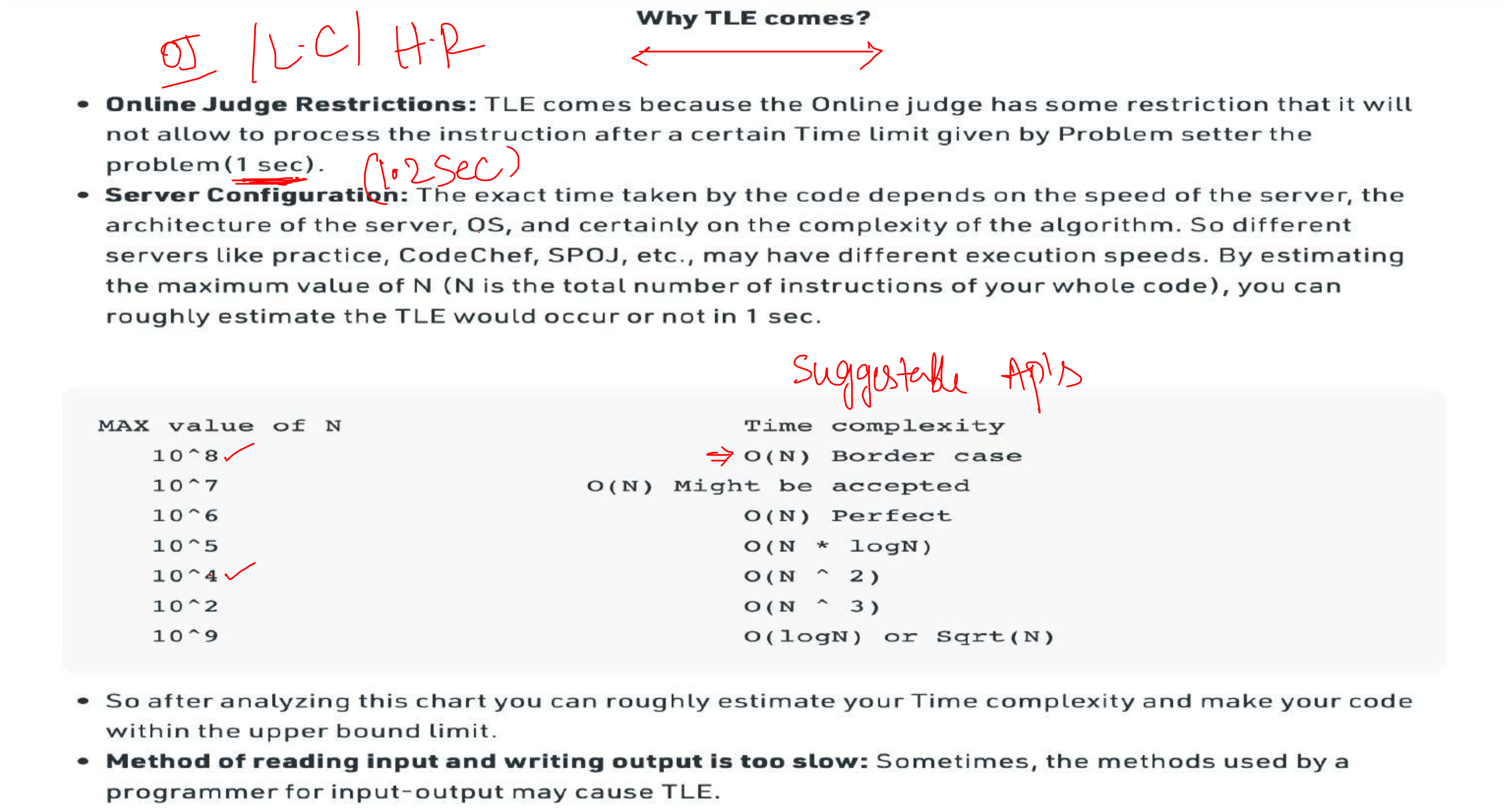
else if

break

contine

return

O(1)



Why TLE comes?

OJ | L.C | H.R

- **Online Judge Restrictions:** TLE comes because the Online judge has some restriction that it will not allow to process the instruction after a certain Time limit given by Problem setter the problem(1 sec). (1.2 sec)
- **Server Configuration:** The exact time taken by the code depends on the speed of the server, the architecture of the server, OS, and certainly on the complexity of the algorithm. So different servers like practice, CodeChef, SPOJ, etc., may have different execution speeds. By estimating the maximum value of N (N is the total number of instructions of your whole code), you can roughly estimate the TLE would occur or not in 1 sec.

Suggestable API's

MAX value of N	Time complexity
10^8 ✓	⇒ O(N) Border case
10^7	O(N) Might be accepted
10^6	O(N) Perfect
10^5	O(N * logN)
10^4 ✓	O(N ^ 2)
10^2	O(N ^ 3)
10^9	O(logN) or Sqrt(N)

- So after analyzing this chart you can roughly estimate your Time complexity and make your code within the upper bound limit.
- **Method of reading input and writing output is too slow:** Sometimes, the methods used by a programmer for input-output may cause TLE.

JS

*Java Script In-built Methods and it's Time Complexity

Mutator Methods.

read

1. push() - O(1) ✓
2. pop() - O(1) ✓
3. shift() - O(n) ⇒
4. unshift() - O(n)
5. splice() - O(n)
6. sort() - O(n log(n))

Accessor methods

1. concat() - O(n)
2. slice() - O(n)
3. indexOf() - O(n)

Iteration methods

1. forEach() - O(n)
2. map() - O(n)
3. filter() - O(n)
4. reduce() - O(n)

U-6 ✓

```
arr=[]
for(i=1; i≤n; i++) → n
{
    arr.shift() => n
}
```

Stack & Queue

DSA + P.L

JS | Java | C++

stack
Queue

Thank
You!