Sprint-2 [Day-2]

Time and Space Complexity-2

ToC $n \geq 1$ → less time, Better ($n > 0$)

$1 \leq N \leq 10^6$

Size of i/p

① $f(n) = n^2$ $g(n) = n^3$	② $f(n) = 2^n$ $g(n) = n^2$	③ $f(n) = 2^n$ $g(n) = 3^n$
$f(n)$: better	2^n n^2 $\log_2 2^n$ $\log_2 n^2$ $n \cdot \underbrace{\log_2 2}_{1}$ $2 \cdot \log_2 n$ n $2 \cdot \log_2 n$ let $n = 2^{1024}$ $2^{1024} >$ $2 \cdot \underbrace{\log_2 2}_{1}^{1024}$ 2048 $g(n)$: best	$2^n < 3^n$ ↳ best

~~$n = -ve$~~

④ $f(n) = n \cdot \log_2 n$

$g(n) = n \cdot \sqrt{n}$

$\cancel{n} \cdot \log_2 n \qquad \cancel{n} \cdot \sqrt{n}$

$$\underbrace{\log_2 n}_{f} < \underbrace{\sqrt{n}}_{g}$$

$\therefore f(n)$: best.

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

$\sqrt{n} > \log_2^n$ $(n>0)$

$\Rightarrow g(n)$: best

⑥ $f(n) = n$ $\qquad$ $g(n) = (\log_2 n)^{100}$

n $\qquad$ $(\log_2 n)^{100}$

$\log_2 n$

f

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

$100 \cdot \log_2 \log_2 n$

g

let $n = 2^{1024}$

$100 \cdot \underbrace{\log_2 \log_2 2^{1024}}_{10}$

1024

1000

$\therefore g(n) : \text{bigt}$

(7) $f(n) = n \cdot \log_2 n$ $\quad \because n > 0$ $\quad g(n) = n^{\log_2 n}$

let $a = 1000$, $b = 10$

$a * b$ $\quad a^b$

$1000 * 10$ $\quad 1000^{10}$

$1000 * 1000 * \cdots$

$f < g$

↳ better.

(8) $f(n) = \sqrt{\log_2 n}$ $\qquad$ $g(n) = \log \cdot \log_2 n$

$(\log_2 n)^{1/2}$ $\qquad$ $\log_2 \cdot \log_2 n$

$\sqrt{n} > \log_2 n$

let $n = 2^{1024}$

$(\log_2 2^{1024})^{1/2}$

$(\underbrace{1024}_{(2^{10})})^{1/2}$

$= \underline{32}$

$= \underbrace{\log_2 \log_2 2^{1024}}$

$= \underline{10}$ ✓

$f > g$

↳ small ⇒ best

Σ, Π, $\$$, Σ

Big-Oh [O( )] : Order of

T.C

↳ unit: Big-oh / order of

B = 130 kcal

$\boxed{O(\ )}$

O( )

↳ Mathematical notation, used to represent T.C

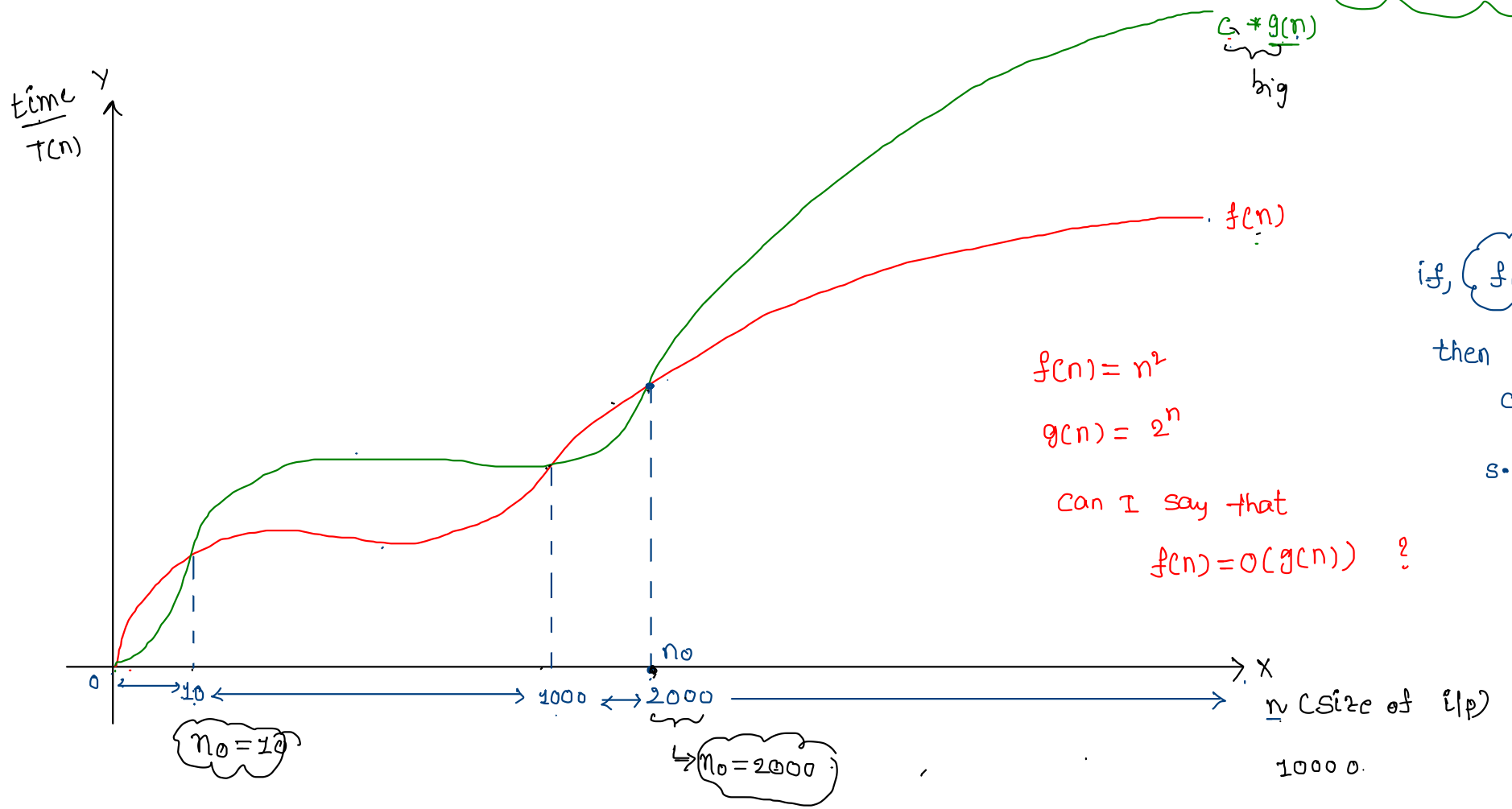
O(n) : order of n

$O(\sqrt{n})$: order of $\sqrt{n}$

$O(n^2)$: order of n^2

O( ) ⇒ used to rep T.C, used bw 2 fn's

T.C = O( ) + matrix function



f(n), g(n)
2 fn's

$f(n) = n^2$

$g(n) = 2^n$

Can I say that

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

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

then

$c * g(n) \geq f(n)$

s.t, $c > 0$ and

$n \geq n_0$

$f(n) = n^2$

$g(n) = 2^n$

Can I say that

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

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

then

$\therefore c * g(n) \geq f(n)$ $\Rightarrow$ $f(n) \leq c * g(n)$

s.t, $c \geq 0$ and

$n \geq n_0$

some +ve const

(+)

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

$n^2 \leq c * 2^n$

$c = 1$; $n^2 \leq 2^n$

some n:

$1 \Rightarrow 1 \leq 2$ ✓

$2 \Rightarrow 4 \leq 4$ ✓

$3 \Rightarrow 9 \leq 8$ ✗

$\rightarrow 4 \Rightarrow 16 \leq 16$ ✓

$5 \Rightarrow 25 \leq 32$ ✓

$6 \Rightarrow 36 \leq 64$ ✓

$n_0 = \cancel{1}$ ✗

≥ 1 ✓

2 ✓

3 ✗

4

$n_0 = 4$

$\Rightarrow$ 5 6 7 . . . .

$\therefore c = 1, \quad n_0 = 4.$

$\therefore f(n) = O(g(n))$ ✓

$f(n) = 2n^2 + 3n + 1 \Rightarrow O(n^2)$ ✓

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

$f(n) \le c \cdot g(n)$, such that

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

$f(n) = 2n^2 + 3n + 1$, $g(n) = n^2$

$2n^2 + 3n + 1 \le c \cdot n^2$

$c = 1000$
$= 10{,}000$

let $c = 1$

$2n^2 + 3n + 1 \le n^2$

$n = 1$: ✗

$n = 2$: ✗

$n = 3$: ✗

$c = 2$

$2n^2 + 3n + 1 \le 2n^2$

$n = \cancel{1}$
$\cancel{2}$
3
$\vdots$

$c = 3$

	f	g	
	$2n^2 + 3n + 1 \le$	$3n^2$	
$n = 1$:	✗		$\rightarrow f$ ✓
$n = 2$:	✗		$\rightarrow f$
$n = 3$:	✗		$\rightarrow f$
$n = 4$:	$45 \le$	48 ✓	$\rightarrow g$
$n = 5$:	$66 \le$	75	$\rightarrow g$
$n = 6$:	$91 \le$	108	$\rightarrow g$
	$\vdots$		

$\therefore$ $c = 3$
$n_0 = 4$
$\therefore f(n) = O(n^2)$

let

$$f(n) = 2n^2 + n^1 + 3\cdot n^0 \Rightarrow 2n^2 \Rightarrow O(n^2)$$

Heighest degree nth term

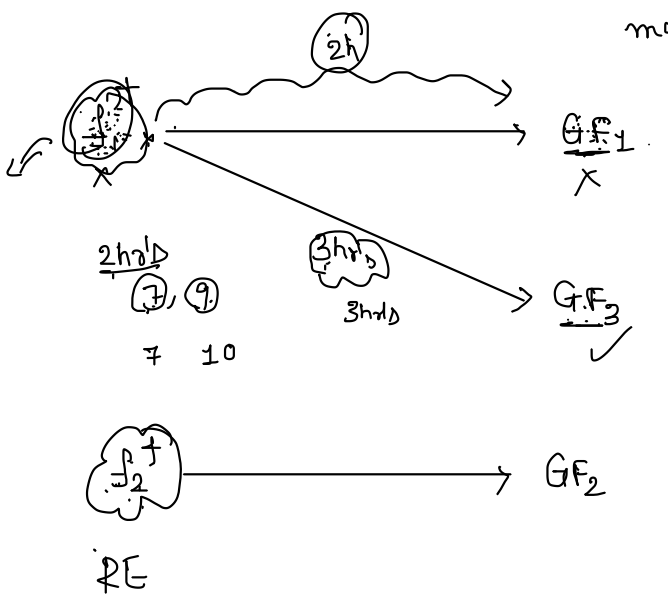
why?

$$\frac{n(n+1)}{2} \Rightarrow \frac{n^2+n}{2} \Rightarrow O(n^2)$$

$$\frac{n(n+1)(2n+1)}{6} \Rightarrow O(n^3)$$

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

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



$\max(2h, 3h) = 3h$

$$f(n) = 3n^4 + 2n^2 + \log_2 n \Rightarrow O(n^4)$$

Code

$$f(n) = O(\underbrace{\sqrt{n}}_{P_1} + \underbrace{\log_2 n}_{P_2}) \Rightarrow \underbrace{O(\sqrt{n})}_{\text{max}}$$

max / Heighest

$$= O(\sqrt{n})$$

$$\Rightarrow O(\log_2 n) \Rightarrow \text{code}$$

O( )

↳ to represent

Approximatly How many times loop running

T.C

```
⇒ for(i=1;i<=n;i++)
  {
      ↳ print(*)
  }
```

T.C ⇒ O(n)

+1

⇒ O(1) constant

n times

break;
Continue;
return
} O(1)

O(n)

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

⇒ n/2

: O(1)

$\Rightarrow \frac{n}{2} = \frac{1}{2} \cdot n$

c

```
i for(i=1; i<=n;i++)  ⇒ n
 {
      j for(j=1;j<=n/2;j++) ⇒ n/2
       {
            c=c+1  : O(1)
       }
 }
```

$$n * \frac{n}{2} = \frac{n^2}{2}$$

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

$$\therefore O(n^2)$$

Nested Loop (loop( ) { loop( ) { } }

↳ Multiply

# of times O.L will run * # of times I.L will run

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

$$n * \frac{n}{4} * n = \frac{n^3}{4}$$

$$= \frac{1}{4} \cdot n^3$$

$$= O(n^3)$$

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

$n * \frac{n}{4} * 1 \Rightarrow O(n^2)$

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

                        c=c+1
                }
        }
}
```

```
i=1
while(i<n)
{
        print(*)
        i=i*2
}
```

```
i=n
while(i>0)
{
        print(*)
        i=i/2
}
```

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

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

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

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

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

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

        i=i/2
}
```

```
for(i=1;i<n;i++)
{
        for(j=1;j<k;j++)
        {
                print("*")
        }

        for(j=1;j<p;j++)
        {
                print("*")
        }
}
```

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

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

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

```
function fun(n)
{
        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
                }
        }
}
```

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

```
function fun(arr,n)
{
    arr.sort()

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

Let T: be the number of test cases

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

Consider the program

```
void function(int n) {
int i, j, count=0;
for (i=n/2; i <= n; i++)
        for (j = 1; j <= n; j = j*2)
                count++;
}
```

The complexity of the program is

1. O(log n)
2. $O(n^2)$
3. $O(n^2 \log n)$
4. O(n log n)

What is the complexity of the following code?

```
i = n
while (i>=1){
    for j = 1 to n
        x=x +1
    i = i/2
}
```

a. $\Theta(n)$
b. $\Theta(\log_2 n)$
c. $\Theta(n/\log_2 n)$
d. $\Theta(n \log_2 n)$

```
function bubbleSort( arr, n)
{
var i, j;
for (i = 0; i < n-1; i++)
{
    for (j = 0; j < n-i-1; j++)
    {
        if (arr[j] > arr[j+1])
        {
        swap(arr,j,j+1);

        }
    }

}
}
```