Date __/__/____

W1 Time and Memory complexity:

Input → [code] → Output

1. Read
2. Understand
3. Ideate → correctness, Efficiency (time limit, Memory limit)
4. Implement
   Debug
5. Get AC

Efficiency

- Best case
- Average case
- Worst case

backend ← [Test file]
Test cases

Time limit is for sum of all the test cases in a test file.

eg:
```
for (int i=0;  i< N; i++) {        // 1+N
    for (int j=0; j<i ; j++) {     // 1 + 2 + 3 + ... + N
        count ++;                  // 1+2+ 3+ ... + N
    }
}
```

$$T(n) = 1 + N + 1 + \frac{N(N+1)}{2} \times 2$$

$$= N^2 + 3N + 2 \longrightarrow O(N^2)$$

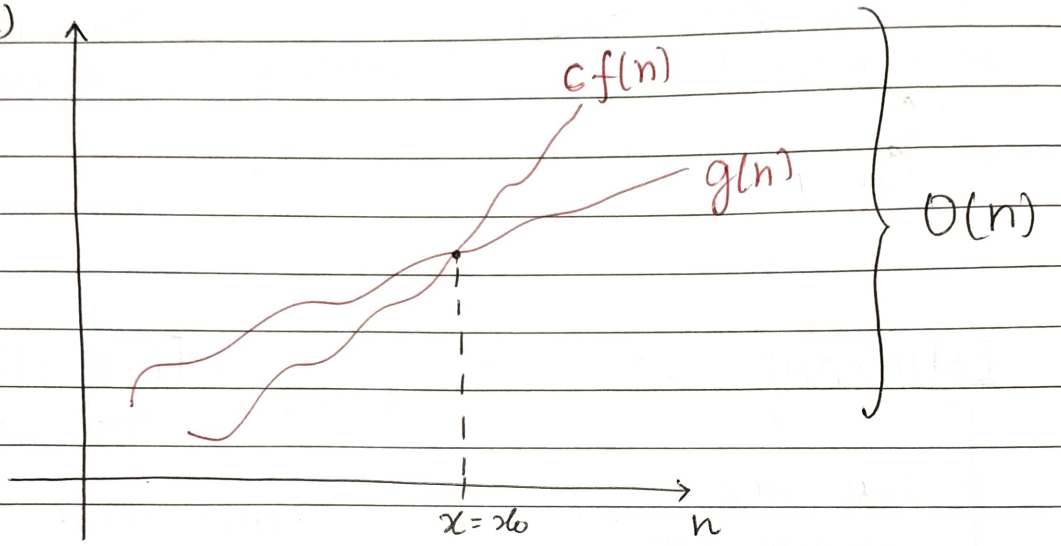
Order Notations

a) $O(f(n))$ → $g(n) \leq c\, f(n)$; $(n \to \infty)$ $\forall\ x > x_0$

b) $\Theta(f(n))$

c) $\Omega(f(n))$ → $g(n) \geq c\, f(n)$

(a)



eg: $g(n) = N^2 + N$

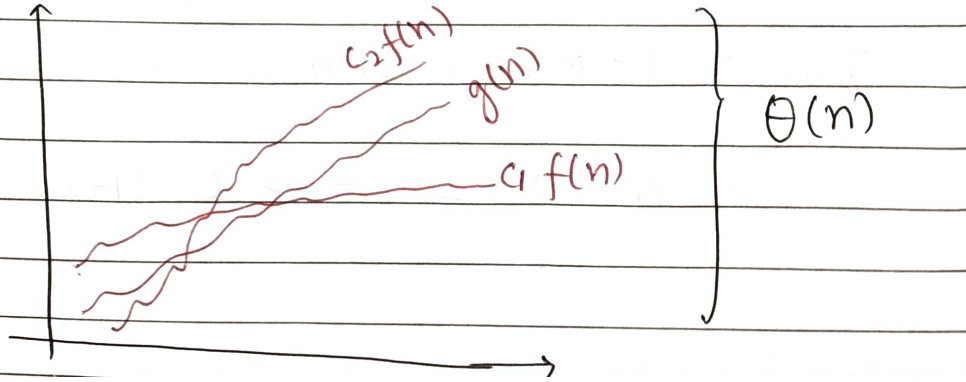
$O(N^2)$ as $2N^2 > N^2 + N$ $(C = 2)$ $[n \to \infty)$

also

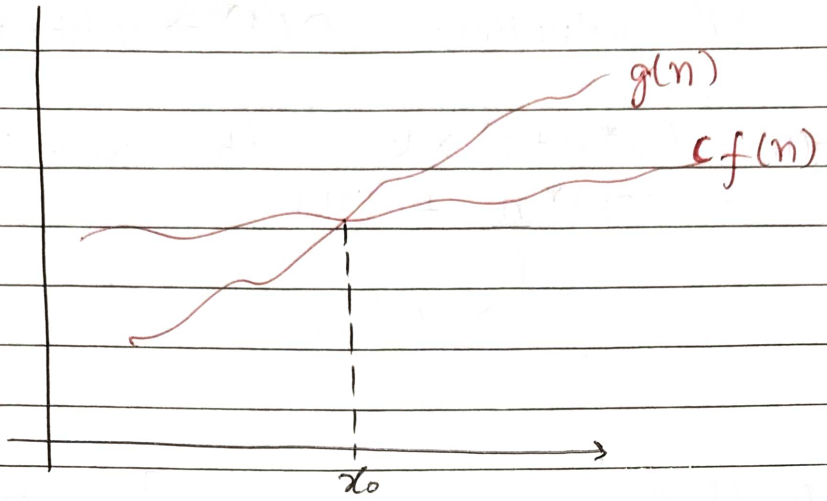
$O(N^3)$ as $N^3 > N^2 + N$ $[n \to \infty]$

→ more tight bound

b) $g(n) = \Theta(f(n))$ if $c_1 f(n) < g(n) < c_2 f(n)$



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



eg: $g(n) = N^3 + N^2$

↳ $\Theta(N^3)$ as $N^3 < g(n) < 2N^3$ ($C_1 = 1, C_2 = 2$)

↳ $\Omega(N^2)$

**

O → upper bound

Ω → lower bound

Θ → exact class (sandwich)

eg: $g(n) = aN^c + \underbrace{B(\sim)}_{\text{lower order term}} + \cdots$

↳ $\boxed{O(N^c)}$, $O(N^{c+1})$, ---

↳ $\Theta(N^c)$

↳ $\Omega(N^c)$, $\Omega(N^{c-1})$, ---

eg 1:

```
for (i = 1; i ≤ 2*n + 7; i++) {
    // 3 instructions      O(1) → constant no. of instr.
}
```

$g(n) = (2*n+7) \times k$ (k = constant)

$= (2k) * n + (7k)$

$n \rightarrow O(n)$

eg 2:

```
for (i=0; i<n; i++) {
    for (j=0; j<n; j++)
    {
        //O(1)
    }
}
```

inner loop → O(n); nesting → multiplying → $O(n^2)$

```
for (int i=0; i< 2*n+5; i++)
{
    //O(1)
}
```

→ O(n)

sequence → addition

$O(N^2) + O(N) = O(N^2)$

dominating

eg 3:

```
for (i → 1 to N) {
    for (j → 1 to N) {
        O(1)
    }
}
```

O(n) → $O(n^2)$

```
for (i → 1 to N) {
    for (j → 1 to M) {
        // O(1)
    }
}
```

O(M) → O(N · M)

$O(N^2) + O(N \cdot M)$
$= O(N(N+M))$

Concept

- $O(f(N,M))$; m, n bound

say $O(N^2)$; $n \leq 10^5$
↓ TL = 1 sec. → $O(10^8)$
$O(10^{10})$

if
$O(N^2) \leq 10^8$ → you can code
$\leq 5 \times 10^7$ → safer

if $O(N^2) \geq 10^9$ → X

eg: $n \leq 10$
then $O(n!)$ or $O(n^7)$ will work

$n \leq 20$
then $O(2^n \cdot n)$ or $O(n^5)$

$n \leq 80 \rightarrow O(n^4)$ will work

$n \leq 400 \rightarrow O(n^3)$ " "

$n \leq 7500 \rightarrow O(n^2)$ " "

$n \leq 10^5 \rightarrow O(N\sqrt{N})$

$n \leq 5 \times 10^5 \rightarrow O(n \log n)$

$n \leq 5 \times 10^6 \rightarrow O(N)$

$n \leq 10^{12} \rightarrow O(\sqrt{N})$ or $O(\sqrt{N} \log N)$

$n \leq 10^{18} \rightarrow O(\log^2 N)$ or $O(1)$ or $O(\log N)$

eg: $f(n) = O(2^n + \log n + N^2 + N^N + c)$

then $f(n) = N^N$

↳ Time = $O(N^N)$

Q1

```
int ans = 0;
int i = 0;
while (i<N) {
    while (i<N && arr[i] != V) i++;
    if (i<N){
        ans++;
        i++;
    }
}
```

linear count
count # V

arr[] → size N
→ What is the Time complexity?

→ no. of statements to be executed depends on i. Each time some statement is executed, i increases.

both loop will break at i = N

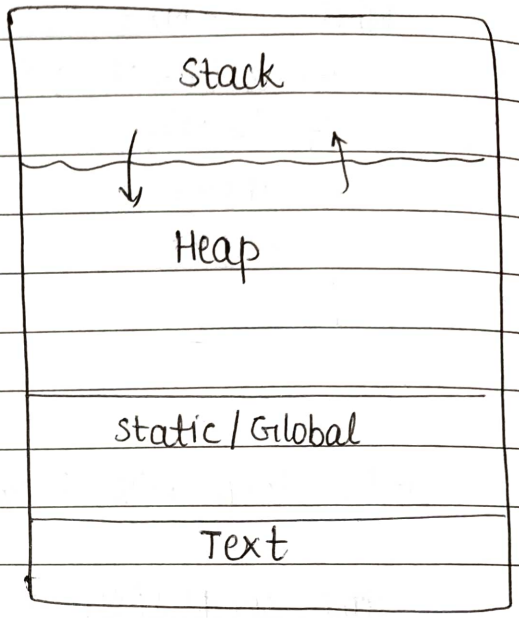
∴

Time complexity = O(n)

Memory Complexity

arr[n][n]
↳ $O(n^2)$

Dont go beyond arr[10^8]
global scope
be safe with arr[10^7]



NP - Hard concept : (not solvable in poly^n time)

↳ $O(N^c)$ ✗
↳ $O(2^n)$ or $O(N^N)$ etc

Master Theorem :-

```
fib(n):
if (n==0||n==1)
{
  return 1;
}
else
{
  return fib(n-1) + fib(n-2)
}
```

O(1)

?? ← here master theorem is not applicable

Say

$$T(n) = a\,T\left(\frac{n}{b}\right) + c$$

for merge sort :

$$T(n) = 2\,T\left(\frac{n}{2}\right) + O(n)$$

Step 1 : Note down a, b, c for

$$T(n) = a\,T\left(\frac{n}{b}\right) + c$$

Step 2 : Calculate $\log_b a$ then $\left(O(n^{\log_b a}), c\right)$

for merge sort $(n^{\log_2 2}, O(n)) \equiv (O(n), O(n))$

Step 3: if $O(n^{\log_b a})$, c is same
then
Time complexity is $c \log n$

else if $O(n^{\log_b a}) > c$
then
Time complexity is $O(n^{\log_b a})$

else
Time complexity is c

for merge sort
Time complexity = $O(n \log n)$

eg 1: $T(n) = 27\, T\left(\frac{n}{3}\right) + O(n^3)$

sol^n: $a = 27,\ b = 3,\ c = O(n^3)$

$\left(O(n^{\log_3 27}),\ O(n^3)\right) \equiv \underbrace{O(n^3),\ O(n^3)}_{\text{same}}$

∴
Time complexity = $c \log n$
= $n^3 \log n$

eg 2: $T(n) = 2\, T\left(\frac{n}{2}\right) + n^2$

a = 2, b = 2, $c = O(n^2)$

$n^{\log_2 2} = n$

$O(n)$, $O(n^2)$ $\Rightarrow$ Time complexity = $O(n^2)$

eg 3: $T(n) = 2T\left(\frac{n}{2}\right) + O(1)$

sol^n: $a = 2, b = 2, c = O(1)$

$\downarrow$

$n^{\log_2 2} = (n)$

$\therefore$

Time complexity = $O(n)$

eg 4: $T(n) = 8T\left(\frac{n}{2}\right) + \frac{n^3}{\log n}$

sol^n: $a = 8, b = 2, c = O\left(\frac{n^3}{\log n}\right)$

$O\ n^{\log_2 8} = O(n^3) > O\left(\frac{n^3}{\log n}\right)$ ✗

$O(n^3)$

Master theorem does not work on logarithmic term.

Ignore log n

$O(n^3)$, $O(n^3)$ $\rightarrow$ ignore $\log(n)$

$\therefore$

Time complexity = $O(n^3 \log n)$

PP1:

```
for(int i = 1 ; i*i <= N ; i++)
{
    for(int j=1 ; j <= i ; j++)
    {
        //O(1) work here
    }
}
```

Soln:

$$= 1 + 2 + 3 + \cdots + \sqrt{N}$$

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

↳ $O(N)$

PP2:

```
for(int i=1 ; i<=N ; i++)
{
    for(int j=1 ; j<=i ; j*=2)
    {
        //O(1)
    }
}
```

Soln:

$$= \log_2(1) + \log_2(2) + \cdots + \log_2(N)$$

$$= \log_2(N!) < \log_2(N^N) \text{ or } N \log N$$

↳ $O(N \log N)$

*

PP3

```
for (int i=1; i<=N ; i++)
{
    for (int j= i ; j<= N ; j+= i )
    {
        //O(1)
    }
}
```

soln:

$i + i + \cdots + i$ (m terms) $= N$

$$m = \frac{N}{i}$$ ← no. of executions of inner loop for some i

$$\text{Time} = N + \frac{N}{2} + \frac{N}{3} + \cdots + \frac{N}{N}$$

$$= N\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{N}\right)$$

$$< N^2$$

↳ actually close to $\log_2 N$

↳ $O(N^2)$

So, more strict bound will be $O(N \log N)$

PP4

```
int power (int a, int b) {
    if (!b)  return 1;

    int temp = power (a, b/2) * power (a, b/2) ;
    if (b%2 == 1)  temp* = a;
    return temp ;
}
```

a^b

Soln: $T(n) = 2T\left(\frac{n}{2}\right) + O(1)$

$n^{\log_2 2} = nO = (1)$ & $O(1)$

same ⇓ $(c \log n)$

∴ Time complexity = $O(\log n)$

$O(n^{\log_2 2}), O(1)$ } $O(n) > O(1)$

∴ Time complexity = $O(n)$

PP5:
```
int power(int a, int b) {
    if (!b) return 1;

    int temp = power(a, b/2);
    temp = temp * temp;
    if (b%2 == 1) temp *= a;

    return temp
}
```

Soln: $T(n) = T\left(\frac{n}{2}\right) + O(1)$

$n^{\log_2 1} = n^0 = 1$ $O(1)$

same ⇒ $c \log n$

Time complexity = $O(\log n)$