16/3/25

Lec-16

Time and Space Complexity

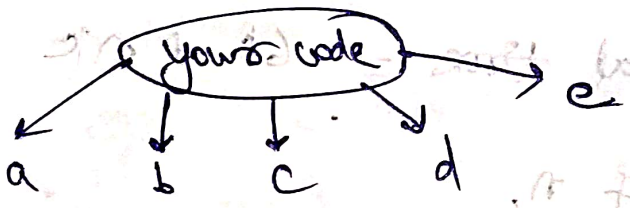
→ What?
How?
Why?

→ Every operation takes some time, eg. int a = 10

- 5ms (in windows)
- 2.5ms (Mac)

Every system can perform operations differently than others computers.

eg. You wrote a piece of code, and you distributed that code via github and 7 friends download your program, every one has diffent type of comp, which has diffent type of specifications and runs program with diffent speed.



we can't tell that total the program is taking 500ms, rather we define it's complexity.

Loops ⇒

eg.

```
int a = 5;      → (K)
int b = 6;      → (K)
int n = 10;     → (K)
for (int i = 0; i < n; i++)      (K) (K) (K)
{
    a = a + b;   → n times
}
```

Assuming that small operations takes time k, which will sum up to $6k$.

% loop will run for n times, ∴ statement written inside loop will run for n times & will take $n \times k$ time.

Adding total time = $6k + nk$

Assuming values of n,

$n = 10$, $6k + 10k$
↓
holds significant value.

$n = 100$, $6k + 100k$
↓
holds less value

$n = 1000$, $6k + 1000k$
↓
holds less value

$n = 10,000$, $6k + 10,000k$

↓ becomes insignificant

adding / subtracting 6 from 10,000 is really insignificant & doesn't make much of the difference.

∴ we don't include those taken by small operations (these are constants) and they take some constant time, for simplicity they take $O(1)$ time.

$10^7 + 3 \approx 10^7$

constant time

Note:- we don't measure constant operations.

Worst case, Average case and best case

Eg:

```
if (marks > 90)
    {
    syso("Excellent");
    }
else if (marks > 80)
    {
    syso("Good");
    }
else if (marks > 60)
    {
    syso("Above Avg");
    }
else if (marks > 40)
    {
    syso("Avg");
    }
else {syso("Try Again");
    }
```

If marks = 95 → (if it, printing excellent).

↓

2 operations

(BEST CASE)

marks = 65

↳ if st.(m > 90)
else if st.(m > 80)
else if st.(m > 60)
printing → Average

→ 4 operations performed (AVERAGE CASE)

marks = 30

↳ if st.(m > 90)
else if st.(m > 80)
else if st.(m > 60)
else if st.(m > 40)
else
printing Fyagam

→ 6 operation performed (WORST CASE)

Worst case → O notation

eg: O(1)
O(n)
O(n²)
O(nlogn)

Average Case → Θ notation

Θ(n)

Best Case → Ω notation

Ω(n)

In Programming, we always measure "WORST CASE"

PREPARE for WORST & HOPE for BEST

Eg:

```
int a=5;
int b=6;
int n=
for(int i=1; i<=(n+n*n); i++)
{
    b=a+1;
    a=b;
}
```

n^3 → $2n^3$

```
for (int i=1; i<=(n*n); i++)
{
    a=b+1;
    a=b;
    b=4;
}
```

n^2 → $3n^2$

```
for(int i=1; i<=n; i++)
{
    a=a+b;
}
```

n → n

total time ⇒ $2n^3 + 3n^2 + n + k$

If $n=3$, $2\times(27) + 3(9) + 3 + k$

$54 + 27 + 3 + k$

$54 + 30$

$84 + k$

~~$n=10$~~

$2\times10^3 + 3\times10^2 + 10 + k$

$2000 + 300 + 10 + k$

(300 is small when compared to 2000)

$n = 10^3$

$2\times(10^3)^3 + 3\times(10^3)^2 + 10^3 + k$

$2\times10^9 + 3\times10^6 + 10^3 + k$

2×10^9 → 100 crore

3×10^6 → 10 lakh

$O(2N^3) \approx O(N^3)$ (10 lakh is small compared to 100 crore)

Time complexity → $O(n^3)$

Q. → for (int i=1; i<=n; i++) {
for (int j=1; j<=n; j++) {
int a = i+j;
}
}

→ $O(n^2)$

---

n=5

i=1, 1<=5 → true

j = ✗ ✗ ✗ ✗ ✗, i++ → (n)

i=2

i=2, 2<=5 → true

j = ✗ ✗ ✗ ✗ 5 → (n)

i=3, i=4, i=5

(n) (n) (n)

n × n times = n^2

Q. → for (int i=1; i<=n; i++)
{
for (int j=1; j<=i; j++)
{
int a = i+j;
}
}

---

i=5

i=1, 1<=5

j=1, j<=1 → 1×1, i++ = 2 — i=1, j → 1 time

i=2, 2<=5

j=1, j<=2 → 2×2, i++ = 3 — i=2, j → 2 time

i=3, 3<=5

j=1, j<=3 → 3×3, i++ = 4 — i=3, j → 3 time

⋮

~~n times~~, ~~O(n) → O(n^2)~~

~~O(n^2)~~

$i=4$, $4<=5$, → $j=1$, $j<=4$ → 4×4 (i, j), $i++=5$ — [$i=4$, $j=4$ times]

$i=5$, $5<=5$ → $j=1$, $j<=5$ → 5×5 (i, j), $i++=6$ — [$i=5$, $j=5$ times]

$i=6$, $6<=5$ → false

total time → $i=1$ ↳ 1 operation

$i=2$ ↳ 2 operation

$i=3$ ↳ 3 operation

$i=4$ ↳ 4 operation

$i=5$ ↳ 5 operation

$n=5$, $1+2+3+4+5$

$n=6$, $1+2+3+4+5+6$

sum of n terms of AP → $\frac{n(n+1)}{2}$

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

$O\left(\frac{n^2}{2}\right) \approx O(n^2)$ TC

Q →
```
int a=0, i=n;
while(i>0){
    a+=i;
    i/=2;
}
```

→ n=3,

a=0, i=3

while(3>0) → true → a = 0+3, a=3
i = 3/2 = 1, i=1

while(1>0) → true → a = 3+1, a=4
i = 1/2 = 0, i=0

---

while(0>0) → false

⇓

2 times → O(1)

→ n=256

i=256

i=128

i=64

i=32

i=16

i=8

i=4

i=2

i=0

9 operations performed

2 operations performed

N N/2 N/4 N/8 N/16 N/32 N/64 N/128 N/256 1

$\frac{N}{2^0}$ $\frac{N}{2^1}$ $\frac{N}{2^2}$ $\frac{N}{2^3}$ $\frac{N}{2^4}$ $\frac{N}{2^5}$ ---------- $\frac{N}{2^x}$

Whenever we are continuously dividing by 2 (or any number),

$$\frac{1}{1} = \frac{N}{2^x}$$

$$2^x = N$$

$\Rightarrow$ (taking log both sides)

$$\log(2)^x = \log(N)$$

$$x \log(2) = \log(N)$$

(taking base 2 both sides)

$$x \log_2(2) = \log_2(N)$$

$$\boxed{\because \log_a(a) = 1}$$

$$\boxed{x \times 1 = \log_2(N)}$$

$\therefore$ Complexity of program is $\boxed{O(\log_2 N)}$

Note:- If something is dividing continuously by $2, 2^2, 2^3, 2^4 \ldots$ or like that or $3, 3^2, 3^3, 3^4 \ldots$, then complexity of program is $\boxed{O(\log N)}$

(*) → base value depend on number which is dividing

$\boxed{O(\log_2 N)}$ $\boxed{O(\log_3 N)}$

Note:-

$N \longrightarrow 1$ ⟺ $1 \longrightarrow N$

divide by x → $\boxed{O(\log_x N)}$ (exponentially decreasing)

multiply by x → $\boxed{O(\log_x N)}$ (exponentially increasing)

Q → for (int i=1; i<n; i++)
{
i*=2;
}

---

n=5

i=1, 1<5

i = 1×2 = 2, i++ = 3

i=3, 3<5

i = 3×2 = 6, i++ = 7

i=7, 7<5 → false

---

n = 32

i=1, 1<32

i = 1×2 = 2, i++ = 3

i = 2×2 = 4

i=3, 3<32

i = 3×2 = 6, i++ = 7

i=7, 7<32

i = 7×2 = 14, i++ = 15

i=15, 15<32

i = 15×2 = 30, i++ = 31

i=31, 31<32

i = 31×2 = 64, i++ = 65

i=65, 65<32 → false

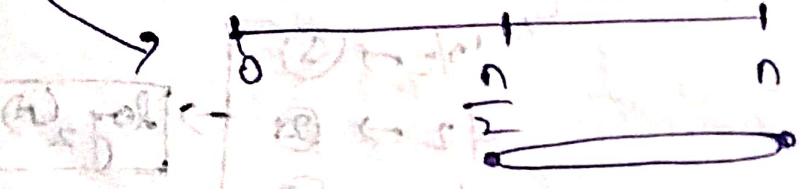
value's ↑↑ exponentially (double)

Time complexity → $O(\log_2 N)$

Eg: for (int i=1; i<n; i++)
{
i *= k;
}

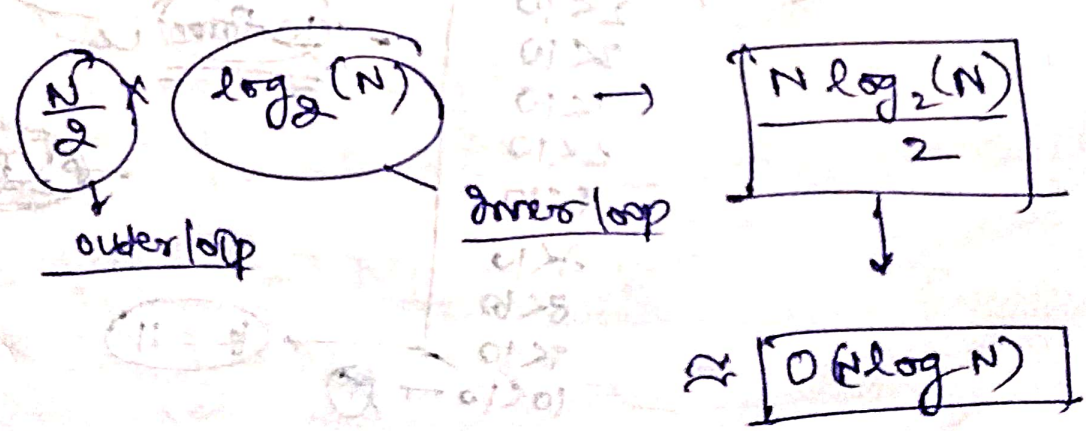
Time complexity → $O(\log_k N)$ Ans.

Q.
```
int i, j, k=0;
for(i=n/2; i<=n; i++){
    for(j=2; j<=n; j=j*2){
        k= k+n/2;
    }
}
```



loop runs from $\frac{n}{2}$ to n that means half way.

and inner loop runs from 2 to n which is somewhat similar to 1 to n, & value of j is increasing exponentially by 2 (i.e $\log_2 N$)



$\simeq O(N \log N)$

Q. →

```
func(int n){
    int k=0;
    for(int i=n; i>0; i=i/2)
    {
        for(int j=0; j<i; ++j)
        {
            ++k;
        }
    }
    sysout(k);
}
```

---

n = 10,

i = 10, i > 0; i = i/2

↳ 10/2 → ① 5/2 → ② 2/2 → ① 1/2 → ⓪ → $\log_2(n)$

Not increasing exponentially

j = 0; j < 10;

i = 1, j = 1

j = 1, 1<10, 2<10, 3<10, 4<10, 5<10, 6<10, 7<10, 8<10, 9<10, 10<10 → ✗ → 10 times

i = 11

n times

$\log_2(n) \cdot n$

$i = N/2$, $j \rightarrow$ $\frac{N}{2}$ times

outer

$i = N/4$, $j \rightarrow$ $\frac{N}{4}$ times

$i = N/8$, $j \rightarrow$ $\frac{N}{8}$ times

⋮

$i = 1$

outer + inner $\rightarrow$ $N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \frac{N}{16} + \cdots$

$$N\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots\right)$$

$$N\left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \cdots\right)$$

↓

sum of GP $\rightarrow$ $\boxed{\frac{a(1-r^n)}{(1-r)}, r<0}$

$$r = \frac{a_2}{a_1} = \frac{\frac{1}{2^1}}{\frac{1}{2^0}} = \frac{1}{2^1} \times \frac{2^0}{1} = \boxed{\frac{1}{2}}$$

$r > 0$

$S_n = \boxed{\frac{a(r^n-1)}{(r-1)}, r>0}$

$\boxed{a=1, r=\frac{1}{2}}$

$$S_n = \frac{1\left(\left(\frac{1}{2}\right)^n - 1\right)}{\left(\frac{1}{2} - 1\right)} = \frac{\frac{1}{2^n} - 1}{-\frac{1}{2}}$$

$$= -\frac{2}{1}\left(\frac{1}{2^n} - 1\right)$$

$N(1 + 0.5 + 0.25 + \cdots\cdots) \approx N(1-2) = \boxed{2N}$

$\boxed{O(2N) = O(N)}$, value will ↓ $(1-2) \rightarrow \boxed{O(N)}$.

Space complexity

→ input space + extra space

```
int n=sc.nextInt();
```

- Hashmap
- Priority queue
- Stack
- Queue
- Array

- If you made array of N size, then comp'xity O(N).

basically, space occupied in the program by data structures.

Note:- In time & space complexity, preference is always given to time.

In online judges, time allowed = 1 sec

⇒ (most compilers do 2^{32} / 10^8 operations in 1 sec.)

Always see constraints, it will hint you time complexity, which will help you to write better & efficient code, within limits.

Note:

$N = 10^5$ → N log(N)

$N = 10^6 \to 10^7 \to 10^8$ → O(N)

$N = 10^9 \to 10^{10} \to 10^{11} \to 10^{12}$ → O(1) / O(log N)