Problem Solving Session – 1

Problem Solving - 1

Space Complexity = O(1)

Q1.

```
ans = 0
for i in range(0, N):        => 3N
    for j in range(0, i+1):  => 3N(N+1)/2
        ans += 1
```

N times

(i+1)

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

Operations

$C_1 + C_2 N + C_3 N^2$

constant op | outer loop | inner loop

i	j	# iterations	j
0	[0, 1)	1-0 = 1	0
1	[0, 2)	2-0 = 2	0, 1
2	[0, 3)	3-0 = 3	0, 1, 2
⋮	⋮	⋮	⋮
N-2	[0, N-1)	N-1	0, 1, .. (N-2)
N-1	[0, N)	N	0, 1, .. (N-1)

$$1 + 2 + 3 + \dots + N = \frac{N(N+1)}{2}$$

① Iterations / Operations $= \frac{N^2 + N}{2}$

$O(N^2)$

$O(N^3)$

$O(N^4)$

$O(N^{100})$

→ Time complexity HW

① ~~Least solved problems~~

② Python least solved

③ Sorting problems.

Mid Module Test

from Wed, 14th sept 6 pm

till next Tue, 20th sept 6pm

Syllabus: Till whatever is covered in Wed, 14th sept class.

(only Intermediate modules)

⇒ Coding + MCQs

Coding → Python / Sorting

MCQs → Time Complexity.

```
for i in ___ :
    for j in range(i):
        ans += i
        for k in ___ :
            ___
```

Big-O an upper bound to the function's time complexity.

$\cancel{O(N^2)}$

```
ans = 0
for i in range(1, N*N*N):      → N^3
    for j in range(1, N*N):    → N^2
        ans += 1
```

$N^3 * N^2 = \boxed{N^5}$ $\Rightarrow \boxed{O(N^5)}$

$1 + 2 + 3 + 4 + 5 <= 5 + 5 + 5 + 5 + 5$

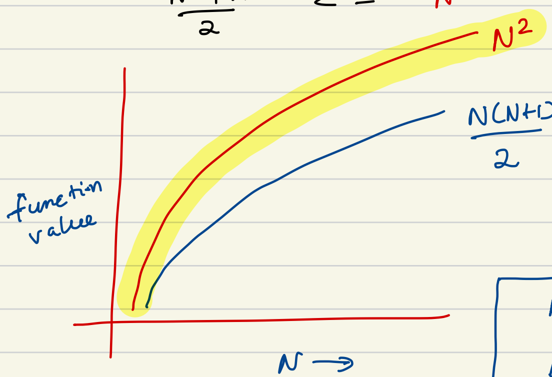
$\frac{5 * \cancel{6}\,3}{\cancel{2}} <= 25 \quad (5 * 5)$

$15 <= 25$

$1 + 2 + 3 + 4 + \dots + N <= \underbrace{N + N + N + \dots + N}_{N \text{ times}}$

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

$\frac{N^2 + N}{2} <= N^2 <= N^3 <= N^4$



$\frac{N(N+1)}{2} = O(N^2) = O(N^3)$

$= O(N^4)$

$---$

A

B

$3 * 10^8$ m/s

i) I was running less than speed of light.

ii) I was running less than speed of Ursain Bolt.

Limitation of Big-O

Cannot compare codes with the same time complexity

$O(n^2)$ $\xrightarrow{?}$ $O(n^2)$

Code A Code B

```
ans = N
for i in range(1, N*N*N):
    for j in range(1, N*N):
        ans //= 2
```

$\boxed{O(N^5)}$

executed N^5 times.

Q2.

⇒

22:10

```
def f():
    ans = 0
    for i in range(1, n+1):
        for j in range(i, n+1, i):
            ans += 1
    print(ans)
```

log n ↑

n log n ↓

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

start = i

stop.

jump = i

i	j	# Iterations.
1	1, 2, 3, 4, --- N	⇒ N/1
2	2, 4, 6, --- N	⇒ N/2
3	3, 6, 9, --- N	⇒ N/3
4	4, 8, 12, 16, -- N	⇒ N/4
.		⋮
N-1		
N	$\boxed{N}$	N/N = 1

⇒ N/i

⇓ Add

$$= \frac{N}{1} + \frac{N}{2} + \frac{N}{3} + \frac{N}{4} + --- \frac{N}{N}$$

$$= N \underbrace{\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + --- + \frac{1}{N} \right)}_{\log N}$$

① Iterations = $\boxed{N(\log N)}$

Operations = $C_1 + C_2 N + C_3 \boxed{N \log N}$

② Time complexity.

= $\boxed{O(N \log N)}$

22:46

Q3. What is the best case, and worst-case time complexity of bubble sort if we are implementing the optimised bubble sort which uses a flag variable for swapping indication?

```
def optimized_bubble_sort( A ):
    n = len(A)
    for i in range(n-1):
        is_sorted = True
        for j in range( n-i-1 ):
            if A[j] > A[j+1] :
                swap A[j] and A[j+1]
                is_sorted = False
        if is_sorted:
            break
    return A
```

I/p : [1, 2, 3, 4, 5] ⇒ $O(N)$ Best case.

I/p : [5, 4, 3, 2, 1] ⇒ $O(N^2)$ Worst case.

22:51

2 min break

Q4.

```
for i in range(1, n):
    for j in range(1, n//4):
        for k in range(1, n):
            break
```

$\Rightarrow O(n)$

$\Rightarrow O(n)$

$\Rightarrow \cancel{O(n)}$ $O(1)$

$\cancel{\boxed{O(n^3)}}$ $O(n^2)$

Identical in no. of operations

```
for i in range(n):
    break
```

? $\Rightarrow O(1)$

$i = 0$

Only $\underline{c}$ operations

$N-1$ times

```
for i in range(1, n):
    for j in range(1, n//4):
        #c operations => constant
```

i	j	# Iterations
1	[1, n//4)	n//4 − 1
2	[1, n//4)	n//4 − 1
3	[1, n//4)	n//4 − 1
⋮	[1, n//4)	⋮
	⋮	
N−2	[1, n//4)	n//4 − 1
N−1	[1, n//4)	n//4 − 1

$\Rightarrow$ (n−1) times.

Total iterations $\Rightarrow$ $(n-1)(n//4 - 1) \simeq n \cdot (n//4)$

$= n^2/4$

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

~~23:10~~
~~23:15~~
Qs.

```
for i in range(n):
    j = i
    while j != 0:
        print(j)
        j = j // 2
```

0 to n−1

⇒ max n

⇒ max $\boxed{\log_2(n)}$

⇒ $\boxed{n \cdot \log_2 n}$

print(1//2) ⇒ 0

⇒ j loop has $\boxed{O(\log_2 i)}$ iterations.

i	j start	j != 0	# iterations
~~0~~	~~0 ⇒ 0 != 0 False~~		~~0~~
2^0 1	1 1 != ⇒ 0 1//2		1 ⇒ +1 $(\log_2 1) = 0 + 1$
2^1 2	2 → 1 → 0		2 ⇒ $(\log_2 2) = 1 + 1$
3	3 → 1 → 0		2 ⇒ $(\log_2 3) = 1 + 1$
2^2 4	4 → 2 → 1 → 0		3 ⇒ +1 $(\log_2 4) = 2 + 1$
5	5 → 2 → 1 → 0		3 ⇒ $(\log_2 5) = 2 + 1$
6	6 → 3 → 1 → 0		3 ⇒ $(\log_2 6) = 2 + 1$
7	7 → 3 → 1 → 0		3
2^3 8	8 → 4 → 2 → 1 → 0		4 ⇒ +1 $(\log_2 8) = 3 + 1$
9	9 → 4 → 2 → 1 → 0		4
10	10 → 5 → 2 → 1 → 0		4
i 11	11 → 5 → 2 → 1 → 0		4 $\boxed{\text{int}(\log_2 i) + 1}$
12	12 → 6 → 3 → 1 → 0		4
13	13 → 6 → 3 → 1 → 0		4
14	14 → 7 → 3 → 1 → 0		4
15	15 → 7 → 3 → 1 → 0		4
2^4 16	16 → 8 → 4 → 2 → 1 → 0		5 ⇒ +1 $= \log_2 16 + 1 = 4 + 1$

int part

Iterations Increasing by 1 when i is a power of 2

No. of times to divide N to set 1 ~~discussed~~ in Time Complexity − 1

```
i = n
while i > 1:
    i = i // 2
```

O(log₂ n) iterations

$\log_2 i$

$n \to n//2 \to n//4 \to --- \to 1 \to 0$

$\Downarrow$

Adding all iterations for j

i is from 0 to $(n-1)$

$$\boxed{(\log_2 1 + 1) + (\log_2 2 + 1) + (\log_2 3 + 1) + \dots + (\log_2 (n-1) + 1)} \Rightarrow$$

$(n-1)$ iterations of i

$$= (n-1) + \underbrace{\boxed{(\log_2 1 + \log_2 2 + \log_2 3 + \dots + \log_2 (n-1))}}_{\Downarrow \; \boxed{S}}$$

$$= O(n) + O(n \log_2 n)$$

$\boxed{S} \Rightarrow$ Big O??

$= O(N \log_2 N)$

$\Rightarrow$ S will always be less than $N \log_2 N$

n values are very large

$n \simeq 10^8$

$\log_2 1 <= \log_2 n$

$\log_2 2 <= \log_2 n$

$\log_2 3 <= \log_2 n$

$\vdots$

$\log_2 (n-1) <= \log_2 n$

$(n-1)$ times

To get an upper bound

$$\boxed{S} <= \boxed{(n-1) \log_2 n} = \underbrace{n \log_2 n}_{\uparrow \text{ Highest power}} - \cancel{\log_2 n}$$

$$O(S) = O(n \log_2 n)$$

Final T.C. = $\boxed{O(n \log_2 n)}$

Q6.

```
for i in range(1, n+1):
    for j in range(1, n+1):
        for k in range(n//2, n+1, n//2):
            c += 1
```

Q7. State True or False for the given time complexity comparison:

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

Please Try Q6 and Q7.
Using the ideas discussed today.

If still not able to solve, will share a recorded video for the two

$9 \to 3 \to 1$

3^4

$27 \to 9 \to 3 \to 1$ $\log_3 27 = 3$

$181 \to 27 \to 9 \to 3 \to 1$ $\log_3 181 = 4$

Doubts

```
i=n
while i>0:
    i=i//2
```

```
i= n
while  i > 0 :
    i = i // 3
```

```
i=n
while i>0:
    i=i // 4
```

- - -

```
i=n
while i>0:
    i=i//x
```

$\log_2 n$ $\log_3 n$ $\log_4 n$ $\log_x n$

Logarithmic Time complexities

$O(\log n)$

by default $= 2$.

Recursion 2 ⟸ Power ⟹ keep multiplying

logarithm ⟹ keep dividing.

⇓

we will build our own power function