ToppersNotes

GATE

COMPUTER SCIENCE & INFORMATION TECHNOLOGY

VOLUME-I

ALGORITHMS

Sierra Innovations Pvt. Ltd.

Contents

Analysing Algorithms 1-33

Divide & Conquer 34-159

Greedy Techniques 160-195

Dynamic Programming 196-232

Graph, Tree & Hashing 233-252

ALGORITHMS

Definition :- It is a combination of sequence of finite steps to solve a particular problem.

example -

a) Subtraction of 2 Nos. :-

```
STN()
{
  i)   Take 2 nos. (a,b)
  ii)  c = a - b
  iii) return (c)
}
```

→ Algorithm is the superset of all Prg. Languages.

→ It may take any I/P or not but definitely gives an O/P as a result.

Properties of An Algorithm :-

1) It should terminate after finite time.
2) It should produce atleast one O/P.
3) It is independent of all prg. languages.
4) Every statement in the algorithm should be deterministic.

(a+b) − c ⇒ Deterministic , a+b−c ⇒ Ambiguous

Steps To Construct Algorithm :-

1) Problem Definition - Knowing the Problem

→ For every possible I/p you must know its O/p (mapping)

I/P	O/P
a	b
c	d
y	z

2) Design Algorithm - Algorithm subject has many algorithms, you have to select 1 suitable Algorithm for your problem.

5 Algo's :- a) Divide & Conquer (DAC)
b) Greedy Algorithm (GD)
c) Dynamic Programming (DP)
d) Back Tracking (BT)
e) Branch & Bound (BB)

3) Flowchart - Diagramatic Representation of Algorithm by showing flow of controls in a sequence (systematic manner).

4) Verification & Testing - For every I/P our O/P must be correct or not (for checking purpose).

5) Coding & Implementation - Selection of language in which coder is suitable for coding (C, C++, Java etc) manually.

6) Analysis :- How much space and how much time it is taking for execution of Algorithm.

a) Space (main memory)] 2 Resources
b) Time (CPU Time)

Program saved in HDD (Hard disk)
Running in MM (Main Memory)
Becoz - Reading instruction from HDD takes more time.

Cost :- Hard disk > Cache Mem. > Main Memory
Speed :- Cache memory > main memory > Hard disk.

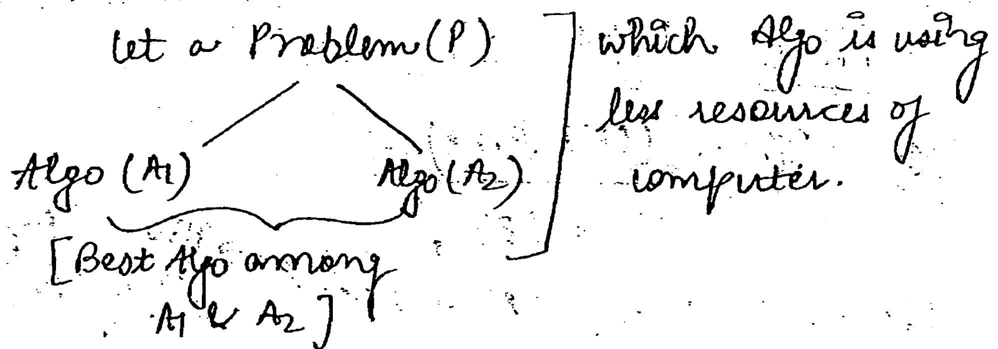
→ Cache memory is faster but more costlier than main memory (medium cost, medium speed).

– Best Algorithm :- Less Time & less space.

⇒ Best 2 steps for an Algo :- Designing & Analysing of Algorithm.

Analysing Algorithm :-

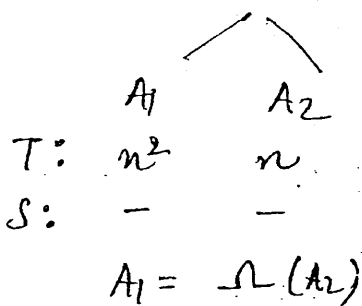
Priority :- (Time > Space)



→ If any problem is having more than 1 solution, best one will be decided by analysis based on 2 things (factors) –

1) Time (C.P.U. Time)
2) Space (Main Memory)

Problem (P)



$A_2 \rightarrow$ Best

$A_2 = O(A_1)$

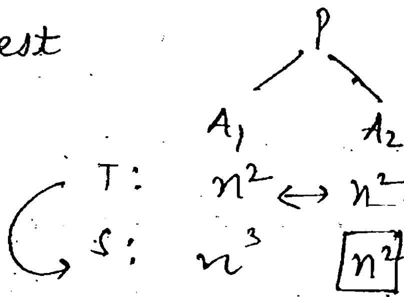
$n = O(n^2)$

↳ Big-oh (Right side Greater)

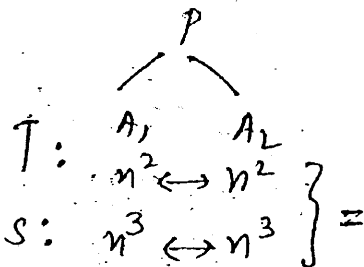
$n^2 = \Omega(n)$

↳ Omega (Right side Smaller)

2nd Priority



$A_2 \rightarrow$ Best



Select any one (A_1, A_2)

Priority: (Time > Space)

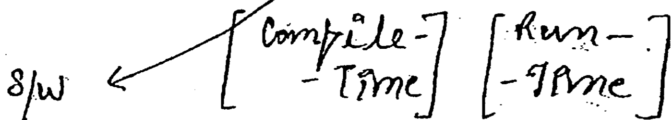
Becoz :- Processor more costlier than main-memory.

TIME COMPLEXITY :-

1) Time (CPU Time) :-

Let a problem (P),

Then, $T(P) = C(P) + R(P)$ → H/W Depends on – (Type of Processor)



S/W Depends on –

(Compiler) → Programming lang. Dependent

↳ Program {C Lang. faster than Java}

Toppersnotes
Unleash the topper in you

TYPES OF ANALYSIS

Apostiary Analysis {Relative Analysis}	APriori Analysis {Absolute Analysis}
1.) Prog. lang. of compiler & Type of Processor (dependent)	1.) P.L. of compiler & Type of Processor (Independent).
2.) System to System answer will change.	2.) Same answer in every system.
3.) Exact Answer → credit goes to processor used. {super-computer}	3.) Approx. Answer (Drawback) → credit goes to programmer. {super-logic}

Apriori Analysis :-

It is a determination of order of magnitude of a statement.
while executing how many times the statement will run ↳ is magnitude (order).

ex 1: - Main()

```
{
i)  x=y+z; ⇒ 1 ⇒ O(1)
}            <constant>
```

Toppersnotes
Unleash the topper in you

ex2:- Main()

```
{
 i) x = y+z;  → 1
 for (i=1; i≤n; i++)
 {
  ii) x = y+z;  → n
 }
}
```

$(n+1) \to O(n)$

ex3:- Main()

$$\left[\begin{matrix} i=1 & i=2 & \cdots & i=n \\ J=1,2,3\cdots n & J=1,2,3\cdots n & \cdots & J=1,2,3\cdots n \end{matrix}\right]$$

```
{
 i) x = y+z;  → 1
 for (i=1 to n)
 ii) x = y+z;  → n
 for (i=1 to n)
 {
  for (j=1 to n)   [n+n+-----n] → n^2   (n-times)
  {
   iii) x = y+z;  → n^2
  }
 }
}
```

$n^2+n+1 \to O(n^2)$

ex4:- Main()

```
{
 for (i=1; i≤100; i++) → 100 → O(1)
}
```

(constant)

⇒ Time Complexity is nothing but finding loops.

⇒ If more loops are there, then find the largest loop where CPU spends more time.

⇒ If there is no loop → O(1)
weather how many statements are there.

*→ cache memory takes valuable things only (i.e. things which CPU wants again & again) → Loop
It is termed as < Locality OF Reference >.

ex 5:- a)

```
Main ()
{
  while (n≥1)
  {
    i) n = n-1   → O(n)
  }
}
```

b)

```
Main ()
{
  while (n≥1)
  {
    i) n = n-2 → n/2 → 1/2 (n) → O(n)
  }
}
```

ex 6:- a)

```
Main()
{ i=1;
  while(i≤n)
  {
    i= i+1; → n → O(n)
  }
}
```

ex 7:- b)

```
Main()
{ i=1;
  while (i≤n)
  {
    i=i+2; → n/2 → O(n)
  }
}
```

ex 8:-

```
Main()
{ while (n≥1)
  {
    n=n-30;
    n=n+15;   → n=n-15; → n/15 → O(n)
  }
}
```

ex 9 :- Main()

```
{
  while (n≥1)
  {
    n = n-30;
    n = n+15;      → n+10;  → infinite loop (It is not an algo).
    n = n+25;
  }
}
```

<already n>

ex 10 :- a) Main()

```
{
  while (n>1)
  {
    n = n/2; → O(log_2 n)
  }
}
```

Proof :- n

$n/2$

$n/2^2$

$n/2^3$

,

,

,

,

$n/2^K = 1$, $n = 2^K$

<Taking $\log_2$ both sides>,

$[\log_2 n = K]$

b) while (n≥1)

```
{
  n = n/2;
}
```

↳ $O(\log_2 n + 1)$ <one step further>

↓

$O(\log_2 n)$

c) while (n>16)

```
{
n = n/2;
}
```

$\frac{n}{2^k} = 16$; $n = 2^k \cdot 2^4$, $n = 2^{(k+4)}$

$(k = \log_2^n - 4) \rightarrow O(\log_2^n)$

ex11:- Main()

```
{
while (n>1)
{
n = n/2;
n = n/3;
}
}
```

→ n = n/6 ; → $O(\log_6^n)$

b) while (n>6)

```
{ n = n/6;
}
```

$\frac{n}{6^k} = 6$

$n = 6^{(k+1)}$

$(k = \log_6^n - 1)$

ex12:- Main()

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

↳ $O(\log_2^n)$

Proof :-

1
2
2^2
2^3
|
$2^k = n$, $k = \log_2^n$

b)

```
i = 15;
while (i<=n)
{
i = 13*i;
}
```

15
13 * 15
$13^2 * 15$
$13^3 * 15$
|
$15 * 13^k = n$

$13^k = n/15$

$k = \log_{13}^{n/15}$ → const. ⇒ $O(\log_{13}^n)$

ex13 :-

```
Main()
{ i=2;
while(i<n)
{
i=i^2;
}
}
```

Proof :-

2

2^2

$2^4 \rightarrow 2^{2^2}$

$2^8 \rightarrow 2^{2^3}$

|

$2^{2^K} = n$

$2^K \log_2 2 = \log_2 n$

$K = \log_2 \log_2 n$

$i=2$

$n=256$

$2 < 256$ ✓

$4 < 256$ ✓

$16 < 256$ ✓

$256 < 256$

$\log_2 \log_2 256$ (inner value 8) $\rightarrow 3$

b)

```
Main()
{
i=2;
while(i<n)
{ i=i^20
}
}
```

Proof :-

2

2^{20}

2^{20^2}

2^{20^3}

|

$2^{20^K} = n$

$\log 20^K = \log_2 n$

$K = \log_{20} \log_2 n$

c)

```
Main()
{
i=13;
while(i<n)
{
i=i^15;
i=i^2;
}
}
```

$\rightarrow i^{30}; \rightarrow O(\log_{30} \log_{13} n)$

Toppersnotes
Unleash the topper in you

ex14:- Main()

```
{
  while (n ≥ 2)
  {
    n = n^(1/2);
  }
}
```

Proof:- n

$n^{1/2}$

$n^{(1/2)^2}$

$n^{(1/2)^3}$

|

$n^{(1/2)^k} = 2$

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

$\frac{1}{2^k} \log_2 n = 1$

$\log_2 n = 2^k$

$$\boxed{\log_2 \log_2 n = k}$$

256 > 2 ✓

16 > 2 ✓

4 > 2 ✓

2 > 2

$\log_2 (\log_2 256) = \log_2 8 \rightarrow \boxed{3}$

b) while (n ≥ 12)

```
{
  n = n^(1/23);   → O(log_23 log_12 n)
}
```

Proof:- n

$n^{1/23}$

$n^{1/23^2}$

|

$n^{\frac{1}{23^k}} = 12$

$\frac{1}{23^k} \log_{12} n = 1$, $\log_{12} n = 23^k$,

$$\boxed{\log_{23} \log_{12} n = k}$$

$\Rightarrow \log_2 n > \log_3 n$

• Small Base → large value

$\log_2 100 > \log_{100} 100 \rightarrow \underline{\underline{1}}$

$\underline{\underline{\approx 7}}$

$\log_2 n > \log_3 n \rightarrow \frac{\log_2 n}{\log_2 3} \approx 1.5$

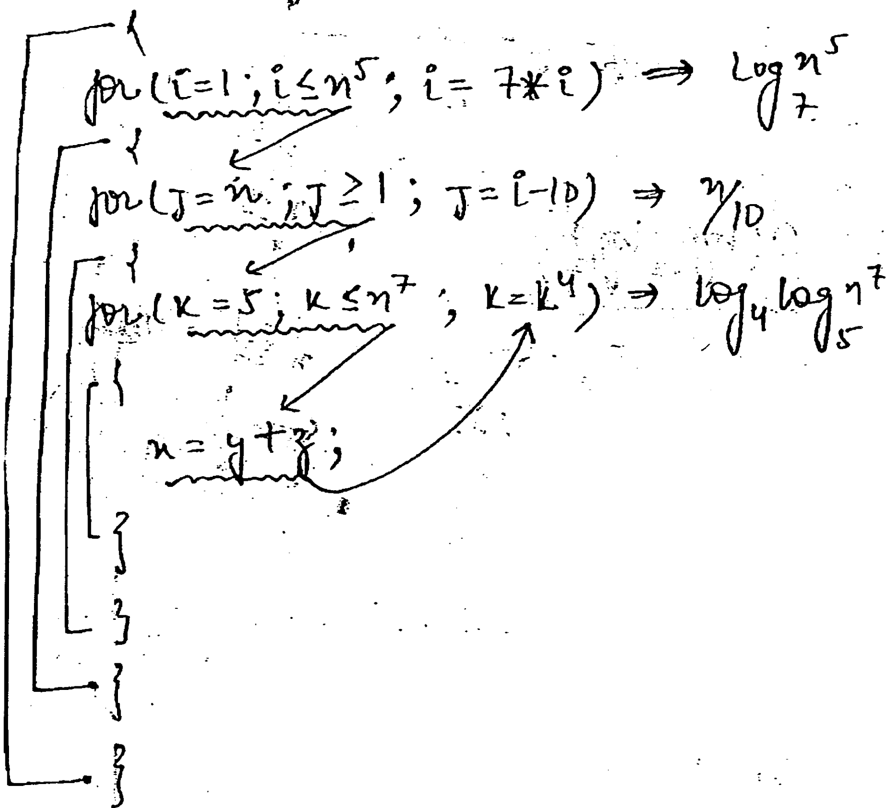
$\log_2 n = 1.5 \log_3 n$ (Mathematically)

<constant neglected>

$\log_2 n = \log_3 n$ (Algorithmically)

\# $\log_b a = \frac{\log_c a}{\log_c b}$

ex15:- Main()



Answer:- $O(\log_4 \log_5 n^7 \cdot \frac{n}{10} \cdot \log_7 n^5)$

⇒ Multiplication because of inner loops (Dependency).

$\Rightarrow \log_4 \log_5 n^7 \Rightarrow \log_4 (7 * \log_5 n)$ $\qquad \log_x ab = \log_x a + \log_x b$

$\log_4 7 + \log_4 \log_5 n$

constant ← (neglect) for $\log_4 7$

final Answer → $O(\log_4 \log_5 n \cdot n \cdot \log_7 n)$

for $(K = 5;\ K \le n^7;$ $K = K^4$, $K = K^{1/2}$ → $K = K^2$

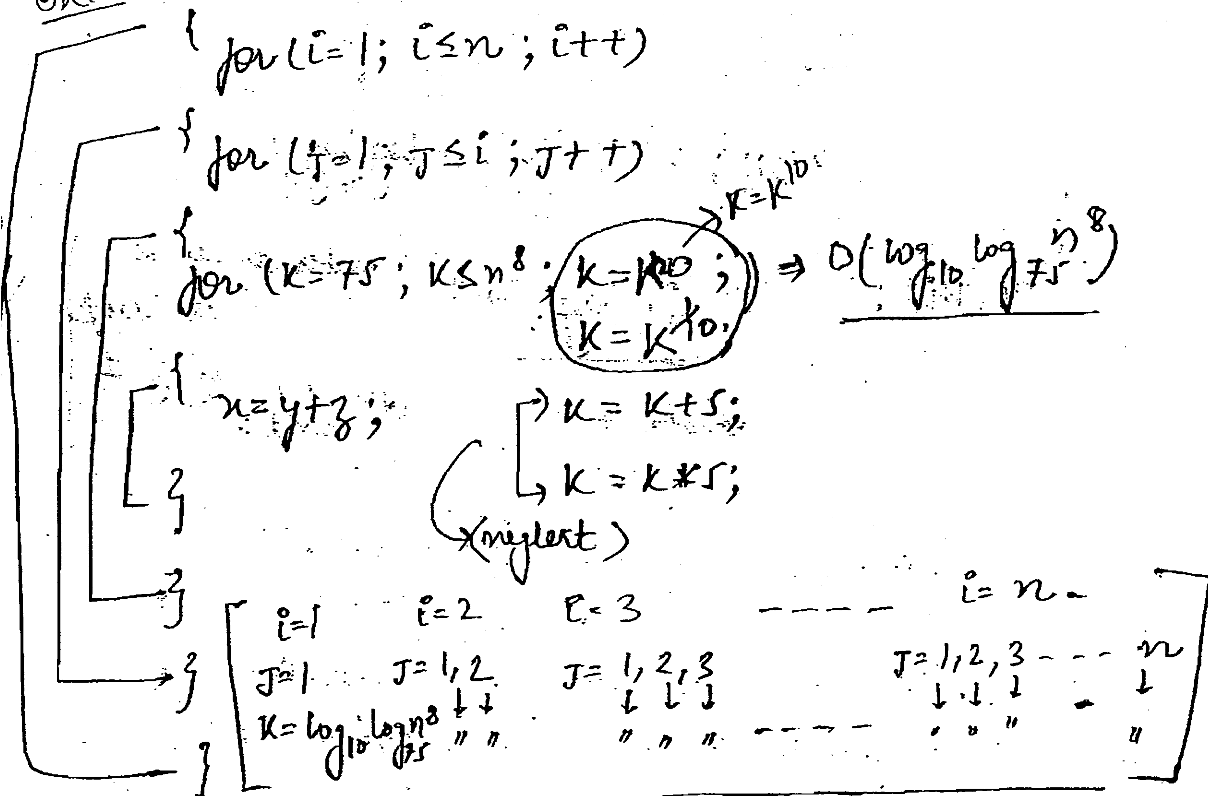
$K = K^{20}$ → $K = K^{40}$

$K = K + 7$, $K = K * 7$ → $K = K^{40}$

{ no effect on the value of K (as small) can be neglected }

$\Rightarrow O(\log_{40} \log_5 n^7)$

ex16:- Main()



```
{ for (i=1; i<=n; i++)
  { for (j=1; j<=i; j++)
    { for (K=75; K<=n^8; K=K^10;)      => O(log_10 log_75 n^8)
                         K=K^{1/10};
      { x=y+z;           K=K+5;   (neglect)
      }                  K=K*5;
    }
  }
}
```

$i=1$	$i=2$	$i=3$	----	$i=n$
$J=1$	$J=1,2$	$J=1,2,3$		$J=1,2,3 --- n$
$K=\log_{10}\log_{75} n^8$	" "	" " "	----	" " " "

$\Rightarrow$ J is totally dependent on i.

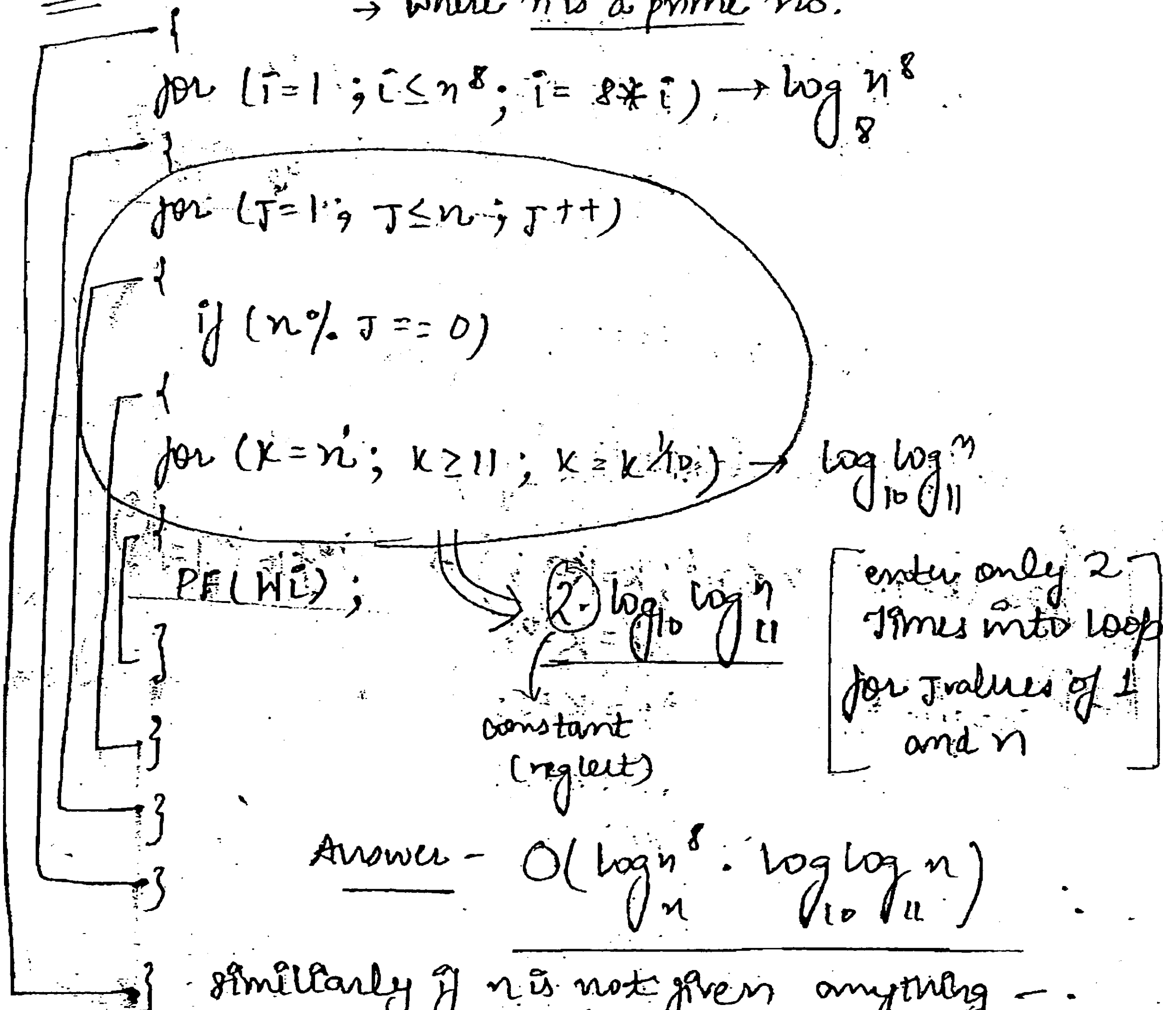
Time complexity (TC) $= \log_{10}\log_{75}^{n} + 2*\log_{10}\log_{75}^{n} + \dots n*\log_{10}\log_{75}^{n}$

$= \log_{10}\log_{75}^{n}\,(1+2+3+\dots n)$

$= \log_{10}\log_{75}^{n}\left(\frac{n(n+1)}{2}\right)$

$= O\left(n^2 \log_{10}\log_{75}^{n}\right)$

ex17 :- Main ()



} similarly if n is not given anything -.
it is for the worst case as even no. then,
$\left(\frac{n}{2}\right)$ times it will enter into loop.

T.. A.. $O\,(\log n^8 \cdot n \log \log n)$

Toppersnotes
Unleash the topper in you

ex18:-

```
Main ()
{
      p=1; q=1;
  for (i=1; i≤n; i*2)
  {
      p++;            → logn
  }
  for (j=1; j≤p; j*2)
  {
      q++;            → log p → loglogn
  }
}
```

$\log n + \log\log n$

Find the value of q → ?

Ans :- $\log\log n$

T.C. = $\log n$ [$\log n > \log\log n$]

ex19:- Main()

```
{
  for (i=1; i≤n; i=i+20)        → n/20
  {
    for (j=n; j≥5; j=j^(1/25))  → log_25 log_5 n <loop>
    {
        p=p+n;                  → n log_25 log_5 n (value of p)
    }
  }
}
```

$n^2 \log\log n$

↑ Every time 'n' is added.

Find the value of p → ?

Ans :- $n^2 \log\log n$

T.C. :- $n \cdot \log_{25}\log_5 n$