ALGORITHMS

1) Difference b/w Algorithms & programs

Algorithm	Program
+ Design	Implementation
+ Domain knowledge	Programmer
+ (Problem knowledge)	Pgmg language
Any language	+ H/w & S/w Dependant
+ H/w & S/w Independent	+ Testing
+ Analyze	

2) Characteristics

1) Input — 0/more
2) output — atleast 1 o/p
3) Definiteness — unambiguous, solvable, clear ($\sqrt{-1}$) ×
4) finiteness — h/w to function / must terminate at some point (Duration)
5) Effectives —

3) How to write an Algorithm

[Datatypes are not decided at alg time]

Algorithm swap (a, b)

Begin {

temp := a;

a := b;

b := temp;

end }

(or)

[variable declaration not consident]

temp ← a

a ← b

b ← temp

CRITERIAS FOR ANALYSIS

(Time in the form of a function)

1) Time - How much time it takes? (Efficiency - faster)
2) space - How much memory it occupies
3) N/w consumption / Data transfer

4) Power Consumption

5) CPU Registers Consumption

Algorithm swap (a, b) — TIME: O(1) — SPACE: O(1)

Begin

temp ← a; → 1

a ← b; → 1

b ← temp; → 1

end

$f(n) = 3$

a → 1

b → 1

temp → 1

$S(n) = 3$ words (Constant)

* Every statement (simple) (or single) → 1 unit of time
* Lengthiness does not change the unit
* Complexity

$x = 5 + a + 6 + b$ → ? 1 (or) 4

(4 statements)

* Constant → O(1) [1 (or) 3 (or) 3000]

FREQUENCY COUNT METHOD

(finding time complexity)

* Algorithm for sum of all elements in an array

Algorithm sum (A, n)

Begin

A = [8 | 3 | 9 | 7 | 2], indices 0 1 2 3 4

n = 5

s ← 0; → 1

for (i ← 0 (1); i < n ($n+1$); i++ (n)) → $n+1$ ($2n+2$)

{

s ← s + A[i]; → n

}

return s; → 1

end

$\boxed{f(n) = 2n + 3}$

$f(n) = 2n+3 \Rightarrow O(n)$ [Degree - 1]

SPACE

Time = O(n)
Space = O(n)

A → n
n → 1
S → 1
i → 1

$S(n) = \underline{n+3} \Rightarrow O(n)$

SUM OF TWO MATRICES

Algorithm Add (A, B, n)

```
Begin
    for (i=0; i<n; i++)                    ---> n+1
    {
        for (j=0; j<n; j++)                ---> n × (n+1)
        {
            c[i,j] = A[i][j] + B[i][j];    ---> n×n
        }
    }
end
```

$f(n) = 2n^2 + 2n + 1$

$O(n^2)$

A, B → Matrices of Dimension $n \times n$

TIME

1) $n+1$ = $n+1$ = $n+1$

2) $n \; (n+1)$ = $n(n+1)$ = n^2+n

3) $n \; (n)$ = n^2 = n^2

$2n^2 + 2n + 1$

SPACE

A → n^2
B → n^2
C → n^2

Scalar { n - 1, i - 1, j - 1 }

$S(n) = 3n^2 + 3$

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

MATRIX MULTIPLICATION

Algorithm Multiply (A, B, n)

```
{
  for (i = 0; i < n; i++)                      → (n+1)
    {                                          → n
      for (j = 0; j < n; j++)
        {                                      → n
          c[i][j] = 0;
          for (k = 0; k < n; k++)              → n
          {                                    → n
             c[i][j] = c[i][j] +
                       A[i][k] * B[k,j];
          }
        }
    }
}
```

TIME COMPLEXITY

$(n+1)$			$= (n+1)$
(n)	$(n+1)$		$= n(n+1)$
(n)	(n)		$= n^2$
(n)	(n)	$(n+1)$	$= n^2(n+1)$
(n)	(n)	(n)	$= n^3$

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

$$f(n) = \boxed{O(n^3)}$$

SPACE

A — n^2	n — 1
B → n^2	i — 1
C → n^2	j — 1
	k — 1

$$S(n) = 3n^2 + 4 = \boxed{O(n^2)}$$

Scanned by CamScanner

(5)

$O(1)$ - constant

$O(\log n)$ - logarithmic

$O(n)$ - Linear

$O(n^2)$ - Quadratic

$O(n^3)$ - cubic

$O(2^n)$, $O(3^n)$, $O(4^n)$ - Exponential

$i=1$; $1<n$ (1) $(n+1)$; $i=i+2$

$n=5$

$i=1$, $1<5$ (T)

$i=3$, $3<5$

$i=5$, $5<5$ (F)

$i=3$, $i=5$

$n=5$

i	j	NOTE
0	0	0
1	0	1

i	j	NOTE
0	0	0
1	0 ✓, 1 ×	1
2	0 ✓, 1 ✓, 2 ✓	2
3	0 ✓, 1 ✓, 2 ✓, 3	3
⋮		
n		

n times

$0 - n$

$i=1$; $i<n$; $i=i+2$

$n=5$

$$\frac{5(5+1)}{2}$$

$$\frac{5\times 4}{2} = 10$$

$$\cancel{5}(30/2 = 15$$

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

$$1+2+3+4+5 = \frac{5(6)}{2}$$

$$15 = 15$$

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

$n = 5$

$i<n$

$1<5$

$3<5$

$5<5$

Scanned by CamScanner

1) for (i = 0; i < n; i++) $\quad n+1$

{

statements; $\quad n$

}

$\quad 2n+1$ $\quad \boxed{O(n)}$

2) for (i = 0; i > 0; i - -) $\quad n+1$

{

Statements; $\quad n$

}

$\quad n+1$ $\quad \boxed{O(n)}$

3) for (i = 1; i < n; i = i + 2) $\quad (n+1)$

{

Statements; $\quad n/2$

}

$\boxed{O(n)}$

4) for (i = 0; i < n; i++) $\quad n+1$

{

for (j = 0; j < n; j++) $\quad n*(n+1)$

{

statements; $\quad n*n$

}

}

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

5) for (i = 0; i < n; i++)

{

for (j = 0; j < i; j++)

{

Statements;

}

}

i	j	NOTE
0	0	0
1	0 ✓ 1 ×	1
2	0 ✓ 1 ✓ 2 ×	2
3	0 ✓ 1 ✓ 2 ✓ 3 ×	3
⋮		⋮
n		n

$1 + 2 + 3 + \dots n = \frac{n(n+1)}{2}$ $\qquad O(n^2)$

$f(n) = \frac{n^2+1}{2}$

6) $P = 0;$
for (i = 1; P <= n; i++)
{
P = P + i;
}

i.	P
1	0+1
2	1+2=3
3	1+2+3=6
4	1+2+3+4=10
⋮	
(k times Execution) repeating	

How many times this will execute?

$1 + 2 + 3 + 4 + \dots + K$

Assume $P > n$ (it stops)

$\therefore P = \frac{K(K+1)}{2}$

$\frac{K(K+1)}{2} > n$ (It will stop) $\qquad O(\sqrt{n})$

$K^2 > n$

$K > \sqrt{n}$

7) for (i = 1; i < n; i = i * 2)
{
statements;
}

i
1
2
$4(2^2)$
$2^3(8)$
2^k

Assume: $i >= n$ $\qquad i = 2^k$

$2^k >= n$

ceil

$2^k = n$

$k = \log_2 n$ $\boxed{O(\log_2 n)}$ $\lceil \log_2 n \rceil$

⑧ for (i=n; i>=1; i=i/2)
{
statements;
}

i
n
$n/2$
$n/2^2$
$n/2^3$
⋮
$n/2^k$ times

Assume $i < 1$,

$n/2^k < 1$

$n = 2^k$

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

⑨ for (i=0; i*i<n; i++)
{
statements;
}

i
0
1
2
3
⋮
n

Assume $i^2 > n$

$i^2 = n$

$i = \sqrt{n} \Rightarrow \boxed{O\sqrt{n}}$

⑩ for (i=0; i<n; i++)
{
statements;
}
n

for (j=0; j<n; j++)
{
statements;
}
n

$\Rightarrow O(n)$

SUMMARY

```
for (i=0; i<n; i++)          -> O(n)
for (i=0; i<n; i=i+2)        -> O(n)
for (i=n; i>1; i--)          -> O(n)
for (i=1; i<n; i=i*2)        -> O(log_2 n)
for (i=1; i<n; i=i*3)        -> O(log_3 n)
for (i=n; i>1; i=i/2)        -> O(log_2 n)
```

ANALYSIS OF IF & WHILE

(1)

```
i=0;                 -> 1 time
while (i<n)          -> n+1 times
{
    statement;       -> n times
    i++;             -> n times
}
```

$3n+2 \Rightarrow O(n)$

(2)

```
a=1;
while (a<b)
{
    statement;
    a=a*2;
}
```

a
1
2
2^2
2^3
⋮
2^k

Terminates when $a >= b$

$2^k >= b$

$2^k = b$

$\boxed{k = \log_2 b}$

$\Rightarrow O(\log n)$

(3)

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

i
n
$n/2$
$n/2^2$
⋮
$n/2^k$

Terminates at $i<1$

$n/2^k < 1$

$n/2^k = 1$

$\Rightarrow O(\log n)$

(4) i = 1;
k = 1;
while (k < n)
{
  statement;
  k = k + i;
  i++;
}

i	k
1	1
2	2
3	4
4	7
5	11
⋮	⋮
m	m

$1+2+3+4+5+\dots m$

$= \frac{m(m+1)}{2}$

Assume $k \geq n$,

$\frac{m(m+1)}{2} \geq n$

$m^2 > n$

$m = \sqrt{n} = O\sqrt{n}$

(5) while (m != n)
{
  if (m > n)
    m = m - n;
  else
    n = n - m
}

m = 6, n = 3

m	n	NOTE
16	2	1
14	2	2
12	2	3
10	2	4
8	2	5
6	2	6
4	2	7
2	2	

⇒ 16/2 (Approx)

⇒ O(n)

GCD of two numbers M & N

$\boxed{\text{Min} \rightarrow O(1) \;\&\; \text{Max } O(n)}$

(6) Algorithm Test (n)
{
  if (n < 5)
  {
    printf("%d", n);
  }
  — O(1)
  else
  {
    for (i=0; i<n; i++)
    {
      printf("%d", i);
    }
  }
  — O(n)
}

COMPLEXITY

Best case → O(1)

Worst case → O(n)

Scanned by CamScanner

TYPES OF TIME FUNCTIONS

1) $O(1)$ → Constant
2) $O(\log n)$ → Logarithmic
3) $O(n)$ → Linear
4) $O(n^2)$ → Quadratic
5) $O(n^3)$ → Cubic
6) $O(2^n)$ → Exponential
7) $O(3^n)$
8) $O(n^n)$

COMPARING CLASSES OF FUNCTIONS

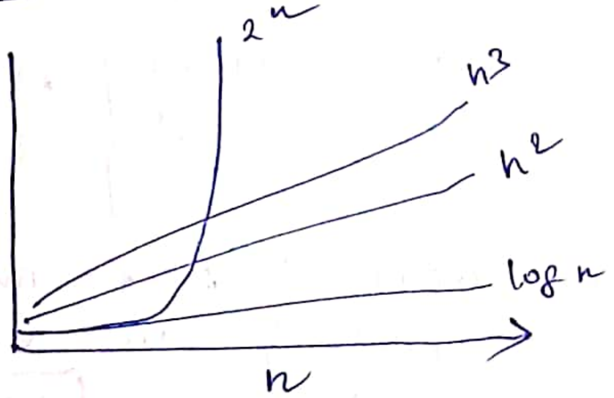
- classes of functions in increasing order of derivatives

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \ldots 2^n < 3^n < \ldots n^n$

$\log n$	n	n^2	2^n
0	1	1	2
1	2	4	4
2	4	16	16
3	8	64	256
3.1	9	81	512

$\log_2 2 = 1$

$\boxed{n^{100} < 2^n}$



Scanned by CamScanner

ASYMPTOTIC NOTATIONS

- Mathematics → functions, Asymptotic Notations to Represents symbol form of a function
  a function
  (or)
  class of a function

- Notations

  a) O → big-oh ⇒ upper bound
  b) Ω → big-omega ⇒ Lower bound
  c) θ → Theta ⇒ Average bound
  (More useful)

  Any function is represent by either of these

(a) BIG-OH NOTATION (O)

DEFINITION :- The function $f(n) = O(g(n))$ iff $\exists$ positive Constants c & n_0 such that

$$f(n) \leq c * g(n) \text{ for all } n \geq n_0$$
(*)

PROOF :-

$$f(n) = 2n+3$$

$$\underbrace{2n+3}_{f(n)} \leq \underset{c}{10}\ \underset{g(n)}{n}$$

[RHS must be greater/Equal to LHS
choose a single term with RHS along with an coefficient]

for all $n \geq 1$ onwards

$$\boxed{\therefore f(n) = O(n)}$$

ALTERNATIVES :-

$2n+3 \leq 2n+3n$
$2n+3 \leq 2n^2+3n^2$
$2n+3 \leq 2^n$

All are Correct since $f(n) \leq c \cdot g(n)$

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

All are True. But we choose the nearest value so closest function is $\boxed{f(n) = O(n)}$

$$\underbrace{1 < \log n < \sqrt{n} <}_{\text{LOWER BOUND}} \underset{\substack{\downarrow \\ \text{AVERAGE BOUND}}}{n} \underbrace{< n\log n < n^2 < n^3 < 2^n < 3^n \ldots < n^n}_{\text{UPPER BOUND}}$$

(b) OMEGA (Ω) NOTATION

DEFINITION :- The function $f(n) = \Omega(g(n))$ iff $\exists$ a constant c & n_0 such that

$$f(n) \geq c \cdot g(n) \quad \forall \; n \geq n_0$$

PROOF :-

$$f(n) = 2n+3$$

$$\underbrace{2n+3}_{f(n)} \geq \underset{c}{1} \cdot \underset{g(n)}{n} \text{ for all } n \geq 1$$

$$\therefore \boxed{f(n) = \Omega(n)}$$

(c) THETA (θ) NOTATION

DEFINITION :- The function $f(n) = \theta(g(n))$ iff $\exists$ a +ve constant c_1, c_2 & n_0 such that,

$$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$$

PROOF :-

$$f(n) = 2n+3$$

$$\underset{c_1 \; g(n)}{1 \cdot n} \leq \underbrace{2n+3}_{f(n)} \leq \underset{c_2 \; g(n)}{5 \cdot n}$$

[g(n) should be the same on both sides]

$$\therefore \boxed{f(n) = \theta(n)} \Rightarrow \text{Average bound of a function}$$

NOTE :- Dont mix asymptotic notations (O, Ω, θ) for worst case, ~~best~~ case & worst case scenarios

EXERCISES FOR ASYMPTOTIC NOTATIONS

(i) $2n^2 + 3n + 4$

(ii) $f(n) = n^2 \log n + n$

(iii) $f(n) = n!$

(iv) $f(n) = \log n!$

PROPERTIES OF ASYMPTOTIC NOTATIONS

(i) GENERAL PROPERTY: if $f(n)$ is $O(g(n))$ then $a * f(n)$ is $O(g(n))$
(True for O, Ω & Θ)

(ii) REFLEXIVE PROPERTY: $(O, \Omega$ & $\Theta)$
if $f(n)$ is given then $f(n)$ is $O(f(n))$
i.e., a function is an upper bound of itself
e.g. $f(n^2)$ then it is $O(n^2)$

(iii) TRANSITIVE PROPERTY (O, Ω, Θ)
if $f(n)$ is $O(g(n))$ & $g(n)$ is $O(h(n))$ then
$f(n) = O(h(n))$

e.g., $f(n) = n$ $\quad g(n) = n^2 \quad h(n) = n^3$
$n = O(n^2) \Rightarrow n^2$ is $O(n^3) \Rightarrow n$ is $O(n^3)$

(iv) SYMMETRIC PROPERTY
if $f(n)$ is $\Theta(g(n))$ then $g(n)$ is $\Theta(f(n))$

e.g., $f(n) = n^2$, $g(n) = n^2$ then
$f(n) = \Theta(n^2)$ & $g(n) = \Theta(n^2)$

* For a fully balanced binary Tree, (17) (15)

COMPARISON OF FUNCTIONS

BEST, WORST & AVERAGE CASE ANALYSIS

(1) LINEAR SEARCH

Search key Element = 7

A	8	6	12	5	9	7	4	3	16	18
	0	1	2	3	4	5	6	7	8	9

i) start scanning from left hand side until it finds the element

ii) Search may be successful / unsuccessful

BEST CASE: If key element is present at first Index (searching key element present at I index)

BEST CASE TIME:- Constant $O(1)$

$$\boxed{B(n) = O(1)}$$

WORST CASE:- Searching a key at last Index

WORST CASE TIME:- n

$$\boxed{W(n) = O(n)}$$

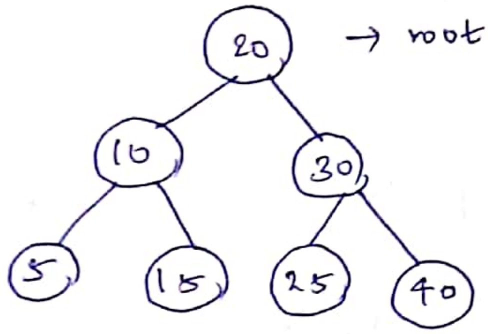
AVERAGE CASE:- $\frac{\text{All possible case time}}{\text{No of cases}}$ (not possible for every Algorithm)

$$\text{Average time} = \frac{1+2+3+\cdots n}{n} = \frac{\frac{\cancel{n}(n+1)}{2}}{\cancel{n}} = \left(\frac{n+1}{2}\right)$$

$$\boxed{A(n) = \frac{n+1}{2}} \Rightarrow O(n)$$

APPLYING ASYMPTOTIC NOTATIONS

② BINARY SEARCH TREE



+ Elements are organized such that for every node
  a) Elements smaller are on LHS
  b) Elements larger are on RHS

eg, Key = 15 (check 20, → NO, (15 < 20) goto LHS
15 < 10 (NO) go to RHS, 15 FOUND)

Time taken = 3 (Equal to height of the tree)

Height of a Binary search Tree = log n

BEST CASE :- searching element at root

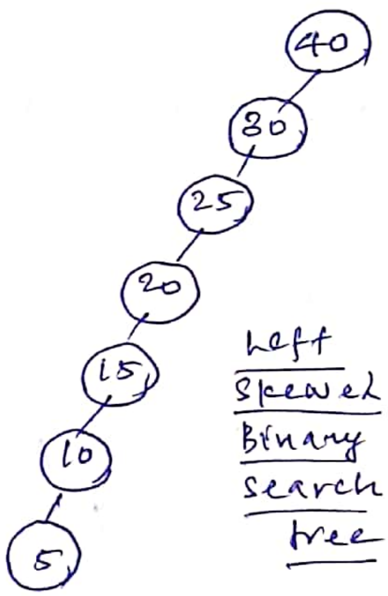
BEST CASE TIME :- B(n) = 1

WORST CASE :- searching for leaf elements

WORST CASE TIME :- Depends on height of a binary search tree

W(n) = h

h can be } min w(n) = log n
max w(n) = n



Left Skewed Binary search tree

height = n

HEIGHT OF BINARY TREE

using log to the base 2 (log 2)

- binary logarithm

* For a fully balanced Binary Tree,

Height	Nodes	Log Calculation
0	1	$\lg_2 1 = 0$
1	3	$\log_2 3 = 1$
2	7	$\log_2 7 = 2$
3	15	$\log_2 15 = 3$

$\log_2 1 = 0$
$\log_2 2 = 1$
$\log_2 3 = 1$
$\log_2 4 = 2$
$\log_2 5 = 2$
$\log_2 6 = 2$
$\log_2 7 = 2$
$\log_2 8 = 3$

NOTE:- In a balanced binary tree the problem size is halved with every iteration. So

$$h = O(\log n)$$

(II) DIVIDE AND CONQUER

* Strategy of solving a problem,
* Strategy : Approach / Design for solving a problem
* Recursive in Nature (Divide & Conquer)

```
DAC (P)
{ if (small (P))
    {
        S(P);
    }
    else
    {
        divide P into P1, P2, P3 ... Pk
        Apply DAC(P1), DAC(P2) .... )
        COMBINE ( DAC(P1), DAC(P2). ...)
    }
}
```

i) Binary search

ii) Finding Max & Min

iii) Mergesort

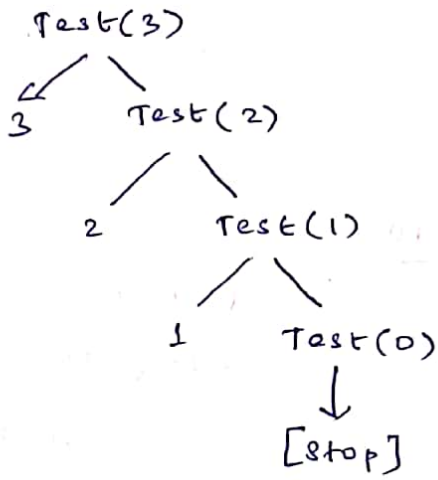
iv) Quick Sort

v) Strassen's Matrix Multiplication

TRACING A RECURSIVE FUNCTION

EXAMPLE – RECURSIVE

```
void Test (int n)
{
    if (n > 0)
    {
        printf ("%d", n);
        Test (n-1);
    }
}
```



TRACING TREE

(OR)

RECURSIVE TREE

TIME FUNCTION :-

printf → Executed 'n' times

Test fn → Executed 'n+1' times

$f(n) = n+1 \Rightarrow O(n)$

$$\boxed{f(n) = O(n)}$$

HOW TO PREPARE A RECURRENCE RELATION ?

+ For recurrence relation function name is T(n)

```
T(n) ←  void Test (int n)
        {
1    ←     if (n > 0)
           {
1    ←        printf ("%d", n);
T(n-1) ←      Test (n-1);
           }
        }
```

$$\boxed{T(n) = T(n-1) + 1}$$

+ Recurrence Relation is

(I)

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1)+1 & n > 0 \end{cases}$$

SOLUTION

GIVEN :- $T(n) = T(n-1) + 1$ → (1)

Substitute $T(n-1)$ in ①

$T(n) = T(n-1) + 1$

$T(n-1) = T(n-2) + 1$

$T(n-2) = T(n-3) + 1$

$T(n) = [T(n-2)+1] + 1$

$T(n) = T(n-2) + 2 \rightarrow$ ②

Substitute $T(n-2)$ in ②

$T(n) = [T(n-3)+1] + 2$

$T(n) = T(n-3) + 3 \rightarrow$ ③

⋮

$T(n) = T(n-K) + K \rightarrow$ ④

Assume $n - K = 0$

$\therefore \boxed{n = K}$

$T(n) = T(n-n) + n$

$T(n) = T(0) + n$

$T(n) = 1 + n \Rightarrow O(n) \Rightarrow \Theta(n)$

(II) RECURRENCE RELATION - DECREASING FUNCTION

```
void Test(int n)                 → T(n)
{
   if (n>0)                      → 1
   {
      for (i=0; i<n; i++)        → n+1
      {
         printf("%d", n);        → n
      }
      Test(n-1);                 → T(n-1)
   }
}
```

$\boxed{T(n) = T(n-1) + 2n + 2}$

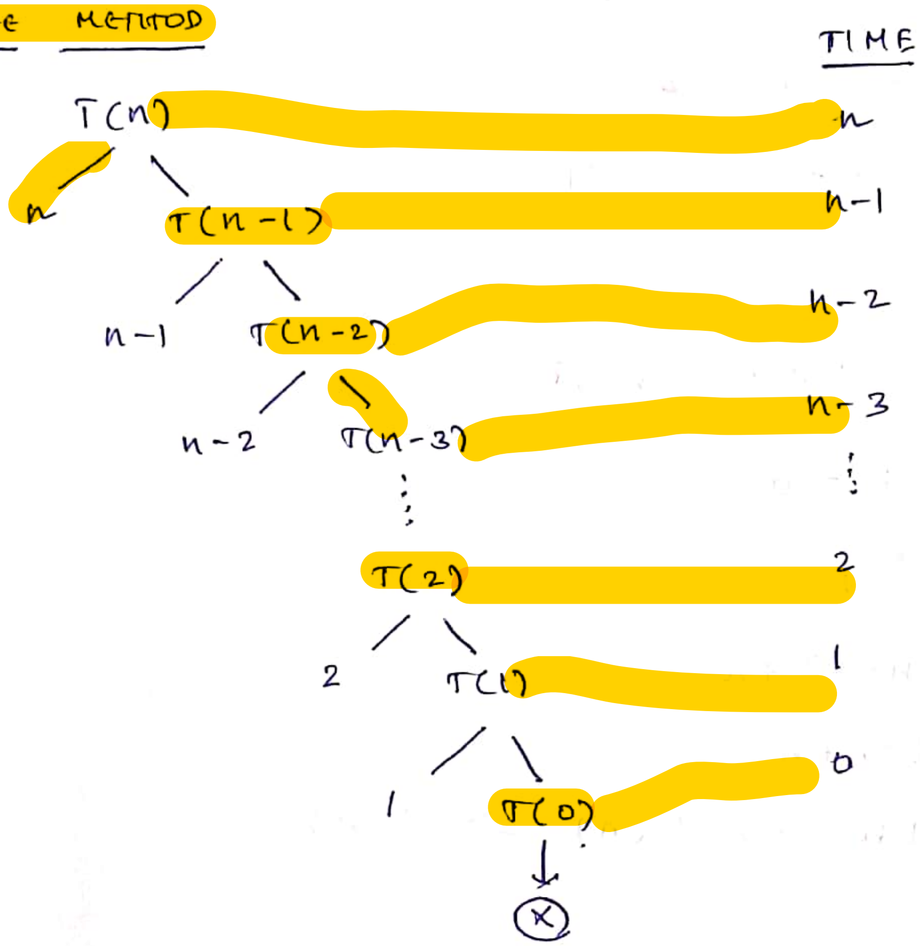
$\Rightarrow \boxed{T(n) = T(n-1) + n}$

The Recurrence Relation is

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + n & n > 0 \end{cases}$$

SOLUTION

(a) TREE METHOD



$$T(n) = 0 + 1 + 2 + \dots + n-1 + n = \frac{n(n+1)}{2}$$

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

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

(B) By Induction (or) Back Substitution Method

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + n & n > 0 \end{cases}$$

Scanned by CamScanner

SOLUTION

$T(n) = T(n-1) + n \rightarrow (1)$

$T(n) = T(n-1) + n$
$T(n-1) = T(n-1) + n - 1$
$T(n-2) = T(n-2) + n - 2$

$T(n) = [T(n-2) + n-1] + n$

$T(n) = T(n-2) + (n-1) + n \rightarrow (2)$

$T(n) = [T(n-3) + n-2] + (n-1) + n$

$T(n) = T(n-3) + (n-2) + (n-1) + n \rightarrow (3)$

⋮

$T(n) = T(n-k) + (n-(k-1)) + (n-(k-2)) + \dots (n-1) + n$

Assume $n - k = 0$

$\therefore \boxed{n = k}$

$T(n) = T(\cancel{n}-\cancel{n}) + (\cancel{n} - \cancel{n}+1) + (\cancel{n} - \cancel{n}+2) + \dots (n-1) + n$

$= T(0) + 1 + 2 + \dots (n-1) + n$

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

$\therefore \boxed{T(n) = O(n^2) \Rightarrow \Theta(n^2)}$

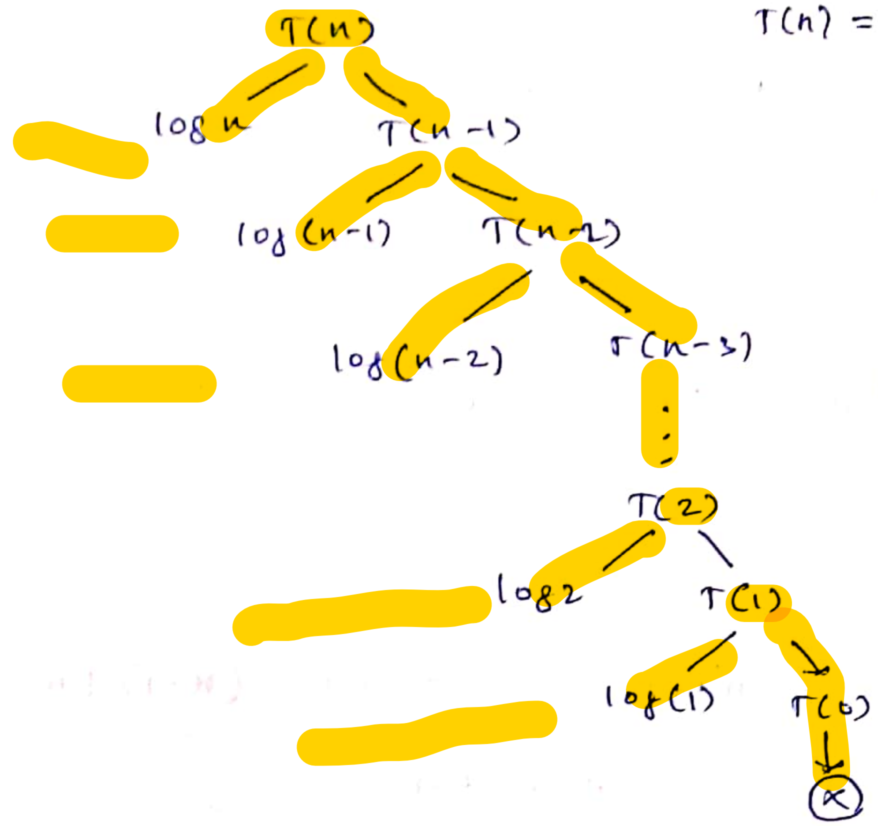
(iii)
```
void Test (int n)
{
  if (n>0)
  {
    for (i=1; i<n; i=i*2)
    {
      printf ("%d", i);     → log n
    }
    Test (n-1);             → T(n-1)
  }
}
```

$\boxed{T(n) = T(n-1) + \log n}$

RECURRENCE RELATION

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n+1) + \log n & n > 0 \end{cases}$$

(a) TREE METHOD



$T(n) = \log n + \log(n-1) + \cdots \log 2 + \log 1$

$= \log [n * (n-1) * \cdots 2 * 1]$

$= \log n!$

(for $n!$ upper bound is n^n)

$\Rightarrow \log n^n$

$\Rightarrow O(n \log n)$

$$\boxed{T(n) = O(n \log n)}$$

(b) INDUCTION / BACK SUBSTITUTION METHOD

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n+1) + \log n & n > 0 \end{cases}$$

SOLUTION:-

$T(n) = T(n-1) + \log n$

$T(n-1) = T(n-2) + \log(n-1)$

$T(n-2) = T(n-3) + \log(n-3)$

$T(n) = T(n+1) + \log n \rightarrow (1)$

Sub. $T(n-1)$ in eqn (1)

$T(n) = [T(n-2) + \log(n-1)] + \log n$

$T(n) = T[n-2] + \log(n-1) + \log n \rightarrow (2)$

Scanned by CamScanner

$$T(n) = [T(n-3) + \log(n-2)] + \log(n-1) + \log n$$

$$T(n) = T(n-3) + \log(n-2) + \log(n-1) + \log n \rightarrow (3)$$

$$T(n) = T(n-k) + \log(n-(k-1)) + \log(n-(k-2)) + \log n$$

$$= T(n-k) \rightarrow (4)$$

Assume $n - k = 0$

$$\boxed{n = k}$$

$$T(n) = T(n-n) + \log(n-n+1) + \log(n-n+2) + \log n$$

$$= T(0) + \log 1 + \log 2 + \dots \log(n-1) + \log n$$

$$T(n) = T(0) + \log n!$$

$$= 1 + \log n!$$

$$\boxed{T(n) \Rightarrow O(n \log n)}$$

SHORTCUTS

i)	$T(n) = T(n-1) + 1$	$\Rightarrow O(n)$
ii)	$T(n) = T(n-1) + n$	$\Rightarrow O(n^2)$
iii)	$T(n) = T(n-1) + \log n$	$\Rightarrow O(n \log n)$
iv)	$T(n) = T(n-1) + n^2$	$\Rightarrow O(n^3)$
v)	$T(n) = T(n-100) + 1$	$\Rightarrow O(n)$ [$n/2$]
vi)	$T(n) = 2T(n-1) + 1$	$\Rightarrow$?

IV

```
Algorithm Test (int n)
{
    if (n > 0)
    {
        printf("%d", n);     → 1
        Test (n-1);          → T(n-1)
        Test (n-1);          → T(n-1)
    }
}
```

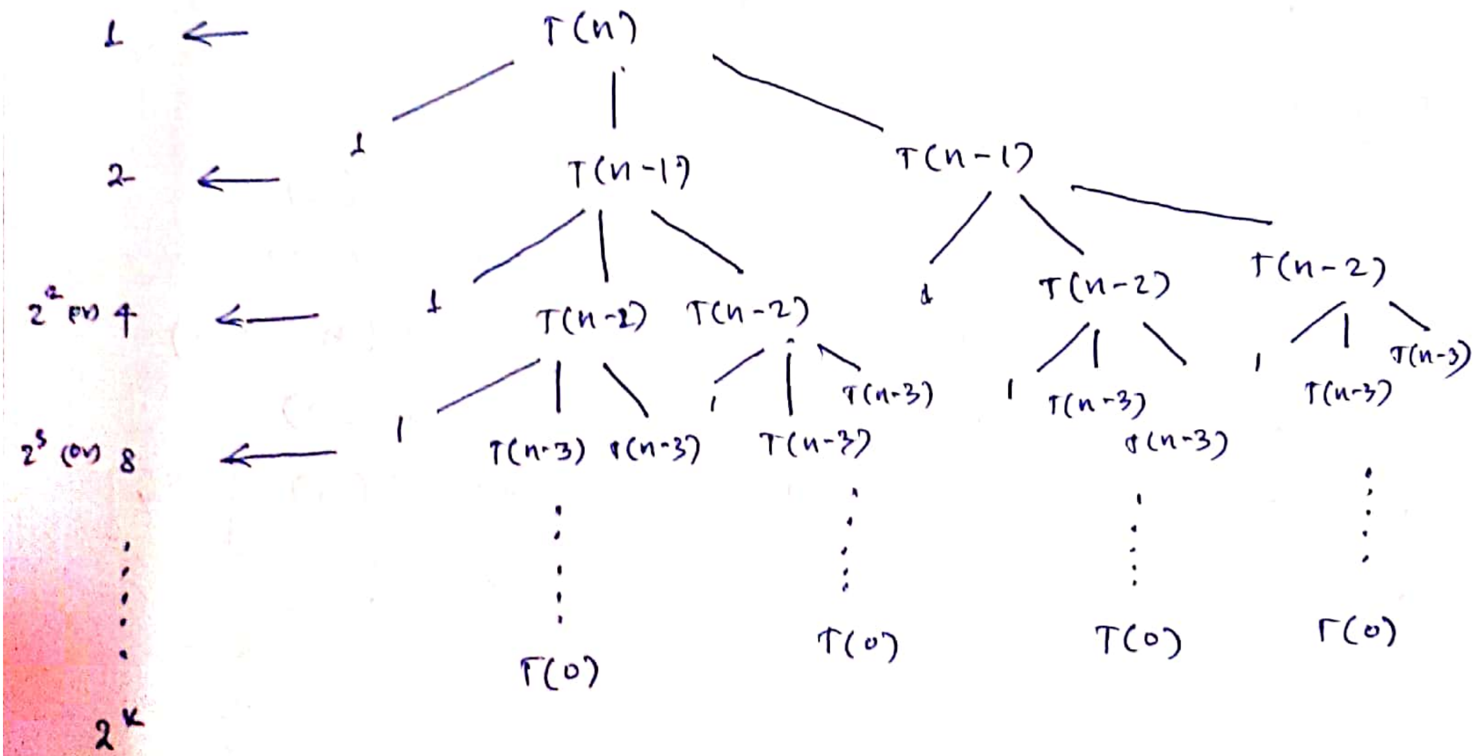
T(n)

$$\boxed{T(n) = 2T(n-1) + 1}$$

RECURRENCE RELATION

$$T(n) = \begin{cases} 1 & n = 0 \\ 2T(n-1)+1 & n > 0 \end{cases}$$

(a) RECURSIVE TREE METHOD



$$1 + 2 + 2^2 + 2^3 + \dots 2^K = 2^{K+1} - 1 \rightarrow (1)$$

(sum of Terms of GP Series)

$$a + ar + ar^2 + ar^3 + \dots ar^K = \frac{a(r^{K+1}-1)}{r-1}$$