Time and space analysis

atlantis
Date
Page

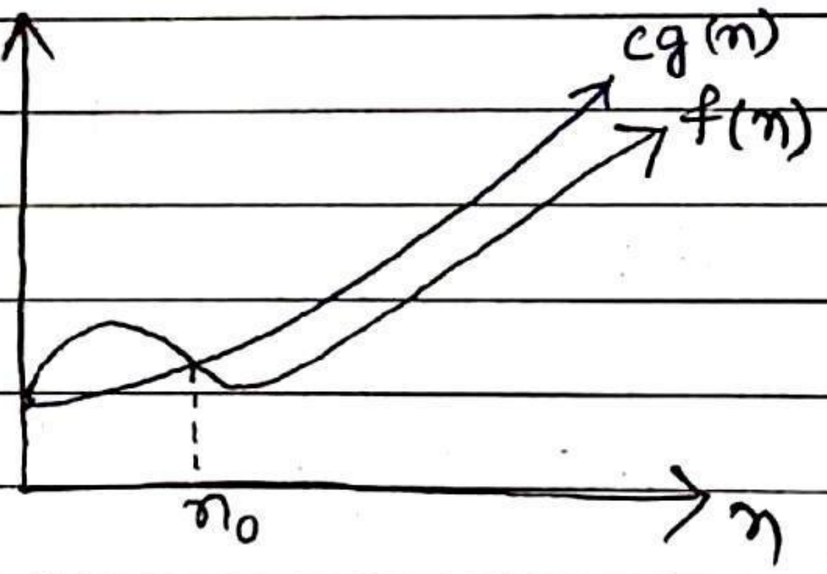
- Introduction to asymptotic notations =

problem
↓
program in C
↓
P_1
A_1 A_2 A_3 A_4 ...

analysis of an algorithm by —
(I) Time (less time)
(II) Memory (less memory)

(i) Big (oh) O :

ex:
$f(n) = 3n+2$, $g(n) = n$
$f(n) = O(g(n))$



$f(n) \leq cg(n)$, $c>0$, $n_0 \geq 1$

$3n+2 \leq cn$.
$c = 4$
$3n+2 \leq 4n$
$\Rightarrow n \geq 2$

$f(n) \leq cg(n)$
$n \geq n_0$
$c>0$, $n_0 \geq 1$.
$f(n) = O\,g(n)$.

So, $f(n) = 3n+2$; $g(n) = n$
$f(n) = O(g(n)) = O(n)$

$g(n) = \{ n, n^3, n^4, \vdots, n^n, 2^n \}$ → always go for least upper bound.
here least upper bound is 'n'.

(ii) Big Omega Ω :

ex:

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

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

$f(n) \geq c g(n)$.

$3n+2 \geq cn$ $\quad c=1$

$n_0 \geq 1$

$\Rightarrow 3n+2 = \Omega(n)$

$f(n) \geq c g(n)$, $n \geq n_0$

$c > 0$, $n_0 \geq 1$.

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

↓

$\log n$

↓

$(\log\log n)$

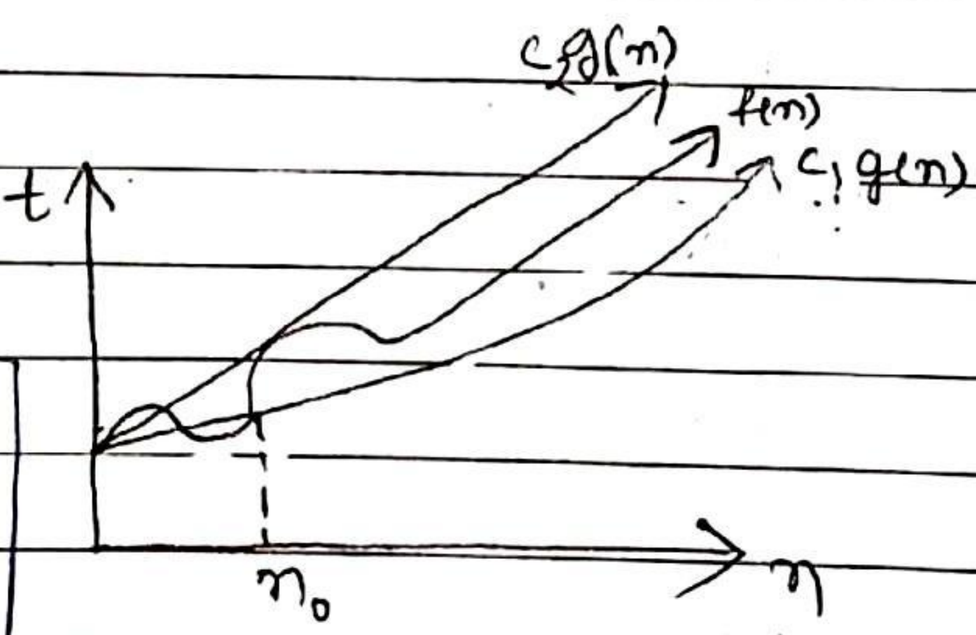
lowers

take always closest ^ bound

here closest bound is 'n'.

(iii) Big theta Θ :

$f(n) = \Theta(g(n))$



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

$c_1, c_2 > 0$

$n \geq n_0$

$n_0 \geq 1$

$3n^2+n+1 = \Theta(n^2)$ (Always take leading terms)

$3n^3+n^2 = \Theta(n^3)$

→ Interested.

O	Ω	θ
↓	↓	↓
Worst case time or the Upper bound.	Best case	Average case

(θ)
→ Average case is used when worst case and best case ~~is same~~. both are same.

ex: array | 5 | 7 | 2 | 3 | 6 | 9 | 20 | 11 | then find x in this array by linear.

→
In Best case time = Ω(1) - (found out in 1st index).
In worst case time = O(n) - (if the size of array 'n' ~~for~~ then found out x in nth position)

In average case time taken = θ(n/2) = θ(n).

• Time Complexity Analysis of iterative programs =
→ Algorithm are two types =

Algo
- Iterative
- Recursive

```
A()
{
  for i=1 to n
    max(a,b)
}
```

```
A(n)
{
  if( )
    A(n/2)
}
```

→ Any program that can be written using iteration could could be written using Recursion.

→ Any program that can be written using recursion could be written using iteration.

→ Any program that not contain iteration and Recursion

→ If there is no iteration and recursion inside the program you need not worry about the time. for such program time — O(1).

Some Example of Iterative program =

① A()
{ int i;
for(i=1 to n)
printf("ravi");
}

→ Time complexity O(n).
(ravi printed n times)

② A()
{ int i, j;
for(i=1 to n)
for(j=1 to n)
printf(ravi);
}

→ Time complexity, O(n²).

③

A()
{ i=1, s=1;
While (s<=n)
{ i++;
s=s+i;
pf("ravi");
}
}

s = 1 3 6 10 15 21 ... n
i = 1 2 3 4 5 6 .... K

Sum of $\frac{K(K+1)}{2} > n$

natural no,

$\frac{K^2+K}{2} > n$

$K = O(\sqrt{n})$

→ Time complexity = $O(\sqrt{n})$.

④

~~A()~~ A()
{ p=1; i=√n
for (i=1; i*i <= n; i++)
pf("ravi");
}

→ Time complexity - O(√n).

⑤

A()
{ int i,j,k,n;
for (i=1; i<=n; i++)
{ for (j=1; j<=i; j++)
{ for (k=1; k<=100; k++)
{ pf("Ravi");
}
}
}
}

→ i=1	i=2	i=3	i=4
j=1 times	j=2 times	j=3	j=4 times
k=100 times	k=2*100 times	k=3*100	k=4*100

i=n
j=n times
k=n*100 th

100 + ~~200~~ 2*100 + 3*100 + 4*100 + 5*100 + ... + n*100

⇒ 100 (1+2+3+4+5+ ..... +n)

⇒ 100 $\frac{n(n+1)}{2}$

⇒ 100 $\left(\frac{n^2+n}{2}\right)$ ⇒ time complexity = O(n²).

atlantis
Date
Page

(6)

```
A()
{ int i,j,k,n;

  for(i=1; i<=n; i++)
  {
    for(j=1; j<= i^2; j++)
    {
      for (k=1; k=n/2; k++)
      { pf
        { pf ("Ravi");
        }
      }
    }
```

→

$i=1$	$i=2$	$i=3$	$i=4$	$i=5$
$j=1$ time	$j=4$	$j=9$	$j=16$	$j=25$ times
$k=n/2*1$	$k=n/2*4$	$k=n/2*9$	$k=n/2*16$	$k=n/2*25$

$i=n$
$j=n^2$ times
$k=n/2*n^2$

→ $n/2 + n/2*1 + n/2*9 + n/2*16 + n/2*25 + \cdots + n/2*n^2$

→ $n/2\,(1+2^2+3^2+4^2+\cdots+n^2)$

→ $n/2\left(\frac{n(n+1)(2n+1)}{6}\right)$ ← A.P.

→ $\frac{1}{12}\,n(n^2+n)(2n+1)$

→ $\frac{1}{12}\,n(2n^3+n^2+2n^2+n)$

→ $\frac{1}{12}\,n(2n^3+3n^2+n)$ → Time Complexity = $O(n^3)$.

$f(n) = n^k + n^{k-1} + \cdots = O(n^k)$

atlantis
Date
Page

10	$(\log_{10} n)$
4	$(\log_4 n)$
3	$(\log_3 n)$

(7)

```
A()
{ for(i=1; i<n; i=i*2)
    pf("ravi");
}
```

→ i = 1, 2, 4, 8, ... n
2^0 2^1 2^2 2^3, ... 2^K

$\frac{n}{2}+1$
$\frac{n+1}{2}$

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

Time complexity or time taken to execute = $O(\log_2 n)$

(8)

```
A()
{
  int i,j,K;
  for(i=n/2; i<=n; i++)   — Indepent loop — n/2
    for(j=1; j<=n/2; j++)   — n/2
      for(K=1; K<=n; K=K*2)   — log2 n
        pf("ravi");
}
```

→ $n/2 * n/2 * \log_2 n$

→ $\frac{n^2}{4} \log_2 n$ → $O(n^2 \log_2 n)$ ⇐ Time complexity.

(9)

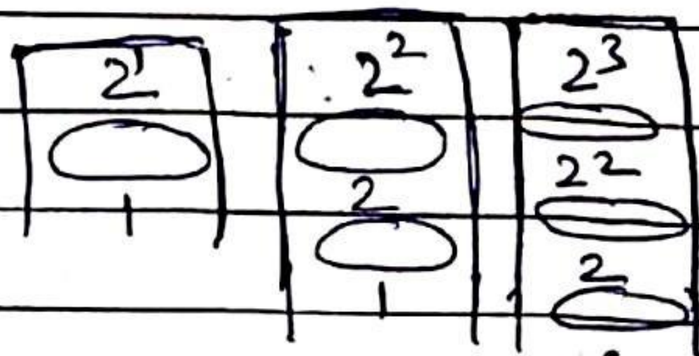
```
A()
{ int i,j,K;
  for(i=n/2; i<=n; i++)   — n/2
    for(j=1; j<=n; j=2*j)   — log2 n.
      for(K=1; K<=n; K=K*2)   — log2 n
        pf("ravi");
}
```

→ $n/2 (\log_2 n)^2$ → $O(n (\log_2 n)^2)$ ← Time complexity
(removed constants)

(10)

assume $n \geq 2$

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



$n = 2^k$

$k = \log_2 n$

3	$\log_3 n$
5	$\log_5 n$
i	

Time complexity $= O(\log_2 n)$.

(11)

```
A()
{ for (i=1; i<=n; i++)
  { for (j=1; j<=n; j=j+i;
  } } { pf("ravi"); }
```

$i=1$	$i=2$	$i=3$	$i=k$
$j=1$ to n	$j=1$ to n	$j=1$ to n ...	$j=1$ to n ...
(Ravi-printed) — n times	$n/2$ times.	$n/3$ times	n/k times

$i=n$
$j=1$ to n
1 times (Ravi printed)

$\rightarrow n + n/2 + n/3 + n/4 + \cdots + n/k + \cdots + n/n$

$\rightarrow n(1 + 1/2 + 1/3 + 1/4 + \cdots + 1/n)$

$\rightarrow n \log n$

$\rightarrow$ time complexity $= O(n \log n)$.

(12)

```
A()
{ Print n = 2^K;
  for (i=1; i<=n; i++)  —— (n)
  {
    j=2
    while (j<=n)   (j = 2^8)
    {
      j = j^2;
      pf("Ravi");
    }
  }
}
```

→ K=1	K=2	K=3
n = 4	$n = 2^4$	$n = 2^8$
j = 2, 4	$j = 2^1, 2^2, 2^4$	$j = 2^1, 2^2, 2^4, 2^8$
n * 2 times	n * 3 times	n * 4 times

→ $\boxed{n*(K+1)}$

→ $n(\log\log n + 1)$

→ ∅ Time complexity of this algorithm — $O(n \log\log n)$.

$n = 2^{2^K}$

$\log_2 n = 2^K$

$K = \log_2 \log_2 n$

atlantis
Date
Page

• Time Complexity Analysis of recursive program =

(1) A(n) ✓
{ if(n>1) ✓
return (A(n-1));
}

By using Back-substitution method you we can find the time complexity of any ~~Algo that~~ recursion program

→ $T(n) = 1 + T(n-1)$; $n>1$
$= 1$; $n=1$

Back Substitution = (method)

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

put equ (2) & (3) into equ (1) =

$T(n) = 1 + 1 + T(n-2)$
$= 1 + 1 + 1 + T(n-3)$
$= 3 + T(n-3)$
⋮
$= K + T(n-K)$ $\quad n-K=1$, $K=n-1$
⋮
$= (n-1) + T(n-(n-1))$
$= (n-1) + T(1)$
$= n-1+1$
$= n$

$\boxed{T(n) = O(n)}$ → Time complexity.

(2)

$$T(n) = n + T(n-1) \quad ; n>1$$
$$= 1 \quad ; n=1$$

Find Time complexity By using back substitution method.

$$T(n) = n + T(n-1) \quad \text{---(1)}$$
$$T(n-1) = (n-1) + T(n-2) \quad \text{---(1)}$$
$$T(n-2) = (n-2) + T(n-3) \quad \text{---(11)}$$

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

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

$$= n + (n-1) + (n-2) + \cdots + (n-(n-2)) + T(1)$$
$$= n + (n-1) + (n-2) + \cdots + 2 + 1 \longrightarrow A.P$$
$$= \frac{n(n+1)}{2} = \frac{n^2+n}{2}$$ (here most significant term n^2)

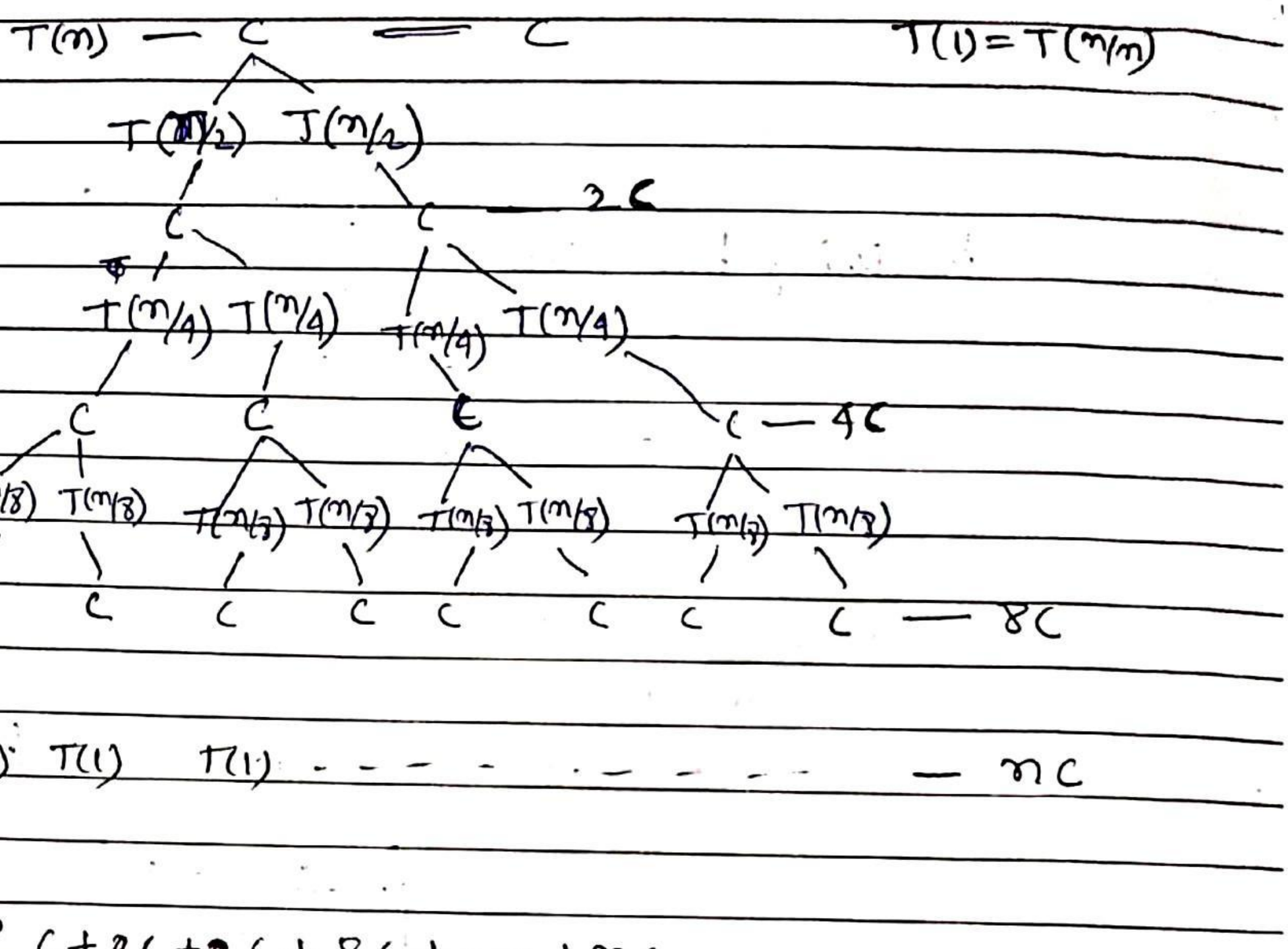
So, time complexity $- O(n^2)$.

(3) Recursion tree method

$$T(n) = 2T(n/2) + C \; ; n>1$$
$$= C \quad n=1$$

Find time Complexity using Recursion tree method -

→ $T(n) = 2T(n/2) + C$.



$\rightarrow C + 2C + 4C + 8C + \cdots + nC$

$\rightarrow C(1 + 2 + 4 + 8 + \cdots + n)$

let $n = 2^K$

$\rightarrow C(2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^K)$ — GP

$\rightarrow C \dfrac{(2^{K+1} - 1)}{(2-1)}$

$\rightarrow C(2^{K+1} - 1)$

$\rightarrow C(2^K \cdot 2 - 1)$

$\rightarrow C(2n - 1)$

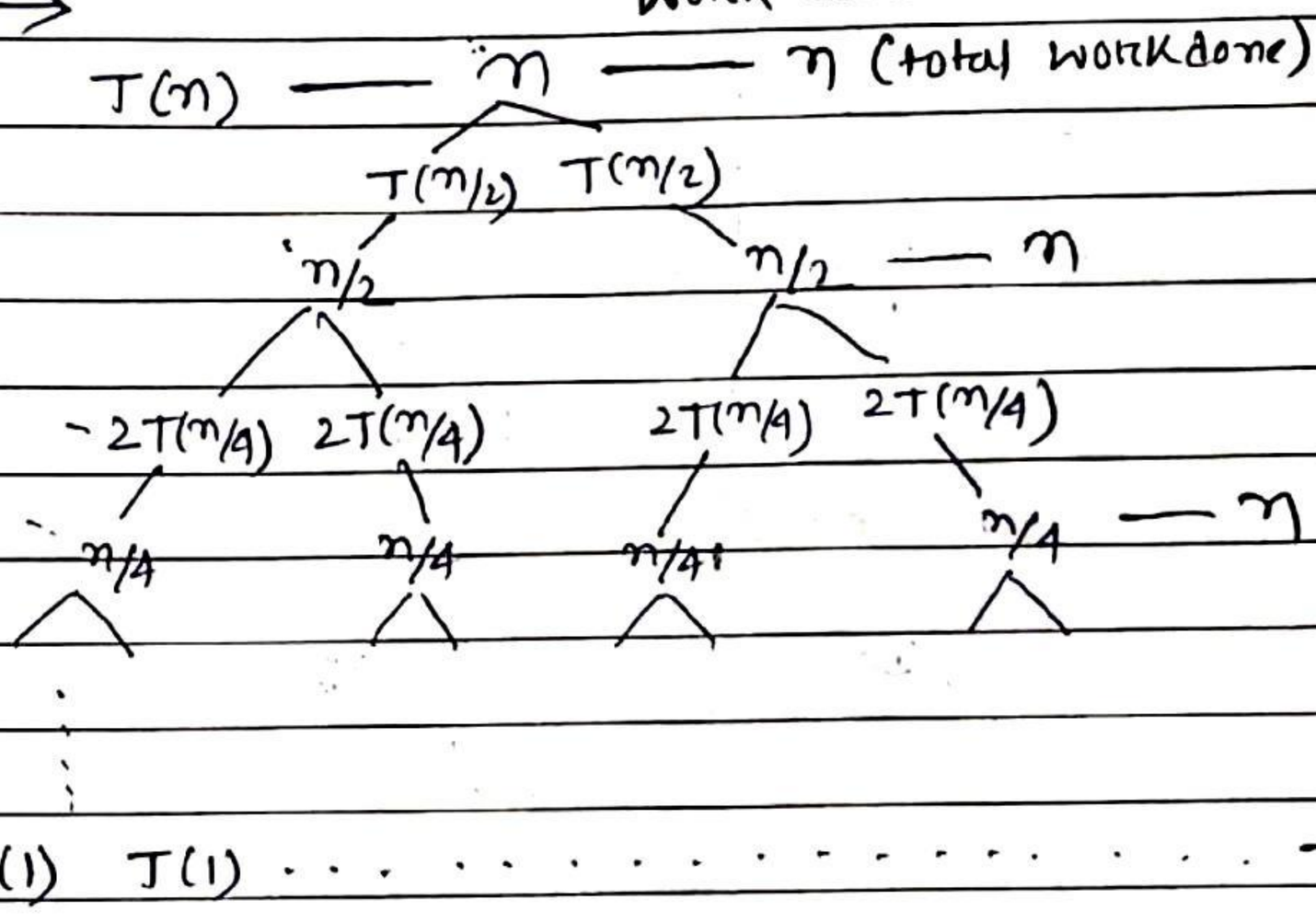
$\Rightarrow O(n)$ → Time complexity.

④ $T(n) = 2T(n/2) + n \; ; \; n \geq 1$

$= 1 \quad ; \; n = 1$

find time complexity using Recursion tree method=

→ Work done each level -



→ $n/2^0 \to n/2^1 \to n/2^2 \to n/2^3 \to \cdots \to \boxed{n/2^k}$ $\boxed{n = 2^k}$ taking log

$k = \log_2 n$

→ $n/2^0 + n/2^1 + n/2^2 + n/2^3 + \cdots + n/2^k$

⇒ $n\left(1/2^0 + 1/2^1 + 1/2^2 + 1/2^3 + \cdots + 1/2^k\right)$

→ $n\,(k+1)$

→ $n\,(\log_2 n + 1)$

So, here, Time complexity is - $O(n \log_2 n)$.

atlantis

Date
Page

~~5 Easiest way to solve recurrence = Master's Theorem = (Time Complexity)~~

- Comparing various functions to analyse time complexity =

(1) ex-1

2^n	n^2
$\Rightarrow \log 2^n$	$\log n^2$
$\Rightarrow n \log_2 2$	$2 \log n_2$
$\Rightarrow n$	$2 \log n$

put, $n = 2^{128}$

$\Rightarrow 2^{128}$	$2 \log 2^{128}$
	$2 * 128$

So, 2^n is larger than n^2.

ex-2 n^2	$n \log n$
$\rightarrow n \times n$	$n \log n$
$\rightarrow n$	$\log n$

function n^2 is greater than $\log n$.

ex-3 n	$(\log n)^{100}$
$\rightarrow \log n$	$\rightarrow 100 \log \log \log n$
$n = 2^{10}$	
$\rightarrow 2^{10}$	$\rightarrow 100 \log \log 2^{10}$
$\rightarrow 1024$	$\rightarrow 1000$

n is larger than $\log n$.

ex-4 $n^{\log n}$ > $n \log n$

$\rightarrow \log(n^{\log n})$ | $\rightarrow \log(n \cdot \log n)$

$\rightarrow \log n \log n$ | $\rightarrow \log n + \log\log n$.

$n = 2^{128}$

$\rightarrow 128 * 128$ | $\rightarrow 128 + 7$

$n^{\log n} > n \log n$

ex-5

$\sqrt{\log n}$ > $\log\log n$

$\frac{1}{2} \log\log n$ | $\log\log\log n$

$n = 2^{2^{10}}$

$\frac{1}{2} * 10$ | $\log 10$

$= 5$ | $= 3.5$

$\sqrt{\log n} > \log\log n$

ex-6

$n^{\sqrt{n}}$ > $n^{\log n}$

$\rightarrow \log n^{\sqrt{n}}$ | $\rightarrow \log n^{\log n}$

$\rightarrow \sqrt{n} \log n$ | $\rightarrow \log n \log n$.

$\rightarrow \sqrt{n}$ | $\rightarrow \log n$

$\rightarrow \frac{1}{2} \log n$ | $\rightarrow \log n \log n$.

$n = 2^{128}$

$\rightarrow \frac{1}{2} * 128$ | $\rightarrow$ [illegible] 7

ex-7

$$f(n) = \begin{cases} n^3 & 0 < n < 10000 \\ n^2 & n \geq 10000 \end{cases}$$

$$g(n) = \begin{cases} n & 0 < n < 100 \\ n^3 & n > 100 \end{cases}$$

→

	0 - 99	100 - 9999	1000 - - - ∞
f(n)	n^3	n^3	n^2
g(n)	n	n^3	n^3

So, $f(n) = O(g(n))$

$g(n) > f(n)$

ex-8 $f_1 = 2^n$, $f_2 = n^{3/2}$, $f_3 = n \log n$, $f_4 = n^{\log n}$

2^n	$n^{3/2}$	$n \log n$	$n^{\log n}$
$n \log 2$	$3/2 \log n$	$\log n + \log \log n$	$\log n \log n$
	$n = 2^{128}$		
$\rightarrow 2^{128}$	$\rightarrow 3/2 * 128$	$\rightarrow 128 + 7$	$\rightarrow 128 * 128$

$f_1 > f_4 > f_2 > f_3$

→ This is how functions are Compared.

$(\log n)^2 \neq \log^2 n$

$\Rightarrow \log n * \log n \neq \log \log n$

atlantis
Date
Page

- **Masters theorem** = (to find time complexity of recursion program easily)

$$\boxed{T(n) = aT(n/b) + \Theta(n^K \log^P n)}$$

$a \geq 1, b \geq 1, K \geq 0$ and p is real Numbers.

i) if $a > b^K$, then $T(n) = \Theta(n^{\log_b^a})$

ii) if $a = b^K$

a) if $P > -1$, then $T(n) = \Theta(n^{\log_b^a} \log^{P+1} n)$

b) if $P = -1$, then $T(n) = \Theta(n^{\log_b^a} \log\log n)$

c) if $P < -1$, then $T(n) = \Theta(n^{\log_b^a})$

iii) if $a < b^K$.

a) if $P \geq 0$; then $T(n) = \Theta(n^K \log^P n)$

b) if $P < 0$; then $T(n) = O(n^K)$.

ex-1

$T(n) = 3T(n/2) + n^2$:

→ after compare with Masters equation —

here $a = 3$, $b = 2$, $K = 2$, $P = 0$

$a \quad < \quad b^K$

$3 \quad < \quad 2^2$

iii) a)

$T(n) = \Theta(n^2 \log^0 n)$

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

Ex-2

$$T(n) = 4T(n/2) + n^2$$

→ After Compare with Masters equation,

$a = 4,\ b = 2,\ K = 2,\ P = 0$

$a = 4$	$b^K = 2^2$

$\boxed{a = b^K}$

ii) → a)

$$T(n) = \theta(n^{\log_2 4} \log^{0+1} n)$$
$$= \theta(n^2 \log n)$$

Ex-3

$$T(n) = T(n/2) + n^2$$

→ After compare with masters Equation -

$a = 1,\ b = 2,\ K = 2,\ P = 0$

$a = 1 \quad b^K = 4$

$\boxed{a < b^K}$

iii) a) $T(n) = \theta(n^K \log^P n)$

$= \theta(n^2)$

Ex-4

$$T(n) = 2^n T(n/2) + n^n$$

↳ In this case masters theorem can't be apply.

Ex-5

$$T(n) = 16T(n/4) + n$$

→ $a=16, b=4, K=1, P=0$

$a=16 \rightarrow b^K=4$

(i) $T(n) = \theta(n^{\log_b^a})$

$= \theta(n^2)$

Ex-6 $T(n) = 2T(n/2) + n\log n$.

→ $a=2, b=2, K=1, P=1$

$a=2 = b^K=2$

$\boxed{a=b^K}$

ii) a)

$T(n) = \theta(n^{\log_b^a} \log^{P+1} n)$

$= \theta(n \log^2 n)$

Ex-7

$T(n) = 2T(n/2) + n/\log n$

$= 2T(n/2) + n \log^{-1} n$.

→ $a=2, b=2, K=1, P=-1$

$\boxed{a=b^K}$

ii) b) $T(n) = \theta(n^{\log_b^a} \log\log n)$

$= \theta(n \log\log n)$

Ex-8

$T(n) = 2T(n/4) + n^{0.51}$.

→ $a=2, b=4, k=0.51, p=0$

$a=2 < b^k = 4^{0.5}$

iii) a) $T(n) = \theta(n^k \log^p n)$

$T(n) = \theta(n^{0.51})$

Ex-9

$T(n) = 0.5T(n/2) + 1/n$ ✗

→ $a=0.5, b=2, k=-1, p=0$ (not possible using master theorem).

$a=0.5$ $b^k = 2^{-1} = 1/2 = 0.5$

$a = b^k$ a should be $a > 1$

iii) ~~a)~~ ~~$T(n) \neq \theta(n^{\log_b a} (\log^{p+1} n)$~~

$= \theta(n^{\log_2 0.5} \log n)$

Ex-10 $T(n) = 6T(n/3) + n^2 \log n$.

→ here, $a=6, b=3, k=2, p=1$

$a < b^k$

iii) a)

$T(n) = \theta(n^k \log^p n)$

$= \theta(n^2 \log n)$

Date
Page

Ex-11

$T(n) = 64\,T(n/8) - n^2 \log n$ ✗

→ here we can't apply masters theorem. ☹ for "−" sign.

Ex-12

$T(n) = 7T(n/3) + n^2$

→ $a=7,\ b=3,\ K=2,\ p=0$

$a < b^K$

iii) a) $T(n) = \theta(n^2)$

Ex-13 $T(n) = 4T(n/2) + \log n$

→ $a=4,\ b=2,\ K=0,\ p=1$

here, $a > b^K$

i) $T(n) = \theta(n^{\log_b a})$

$= \theta(n^2)$

Ex-14

$T(n) = \sqrt{2}\,T(n/2) + \log n$

$a=\sqrt{2},\ b=2,\ K=0,\ p=1$

here,

$a > b^K$

i) $T(n) = \theta(n^{\log_b a})$

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

$= \theta(\sqrt{n})$

Ex-15

$$T(n) = 2T(n/2) + \sqrt{n}$$

→ $a=2,\ b=2,\ k=1/2,\ p=0$

here, $\boxed{a > b^k}$

i) $T(n) = \theta(n^{\log_b a})$

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

$T(n) = \theta(n)$

Ex-16

$$T(n) = 3T(n/2) + n.$$

→ $a=3,\ b=2,\ k=1,\ p=0$

here, $\boxed{a > b^k}$

i) $T(n) = \theta(n^{\log_2 3})$

Ex-17

$$T(n) = 3T(n/3) + \sqrt{n}.$$

→ $a=3,\ b=3,\ k=1/2,\ p=0$

here, $\boxed{a > b^k}$

i) $T(n) = \theta(n^{\log_3 3})$

$= \theta(n).$

Ex-18

$$T(n) = 4T(n/2) + cn.$$

$\Rightarrow$ $a=4,\ b=2,\ k=1,\ p=0$

here, $\boxed{a > b^k}$

i) $T(n) = \theta(n^{\log_2^4})$

$= \theta(n^2)$.

Ex-19

$$T(n) = 3T(n/4) + (n\log n).$$

$\rightarrow$ $a=3,\ b=4,\ k=1,\ p=1$

here, $\boxed{a < b^k}$

iii) a)

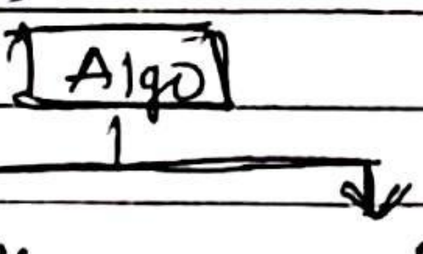
$T(n) = \theta(n^k \log^p n)$

$= \theta(n^1 \log^1 n)$

$= \theta(n\log n)$.

atlantis
Date
Page

Analysis space complexity of iterative and recursive Algorithm —



- Space Complexity for Iterative program =

(1) Algo (A, 1, n)
{
int i; ——— (1)
for (i=1 to n)
A[i] = 0;
}

→ Space complexity = O(1)

Algo (A, 1, n)
{
int i, j; ——— (2)
for (i=1 to j)
A[i] = 0;
}

→ Space complexity = O(1)

(2) Algo (A, 1, n)
{
int i; ——— (n+1)
creat B[n]; ——— (n)
for (i=1 to n)
B[i] = A[i];
}

→ Space complexity = O(n)

(3) Algo (A, 1, n)
{ creat B[n, n] —— n^2
int i, j; —— 2 (n^2+2)
for (i=1 to n)
for (j=1 to n) → here space complexity = $O(n^2)$
B[i, j] = A[i]
}

atlantis
Date
Page

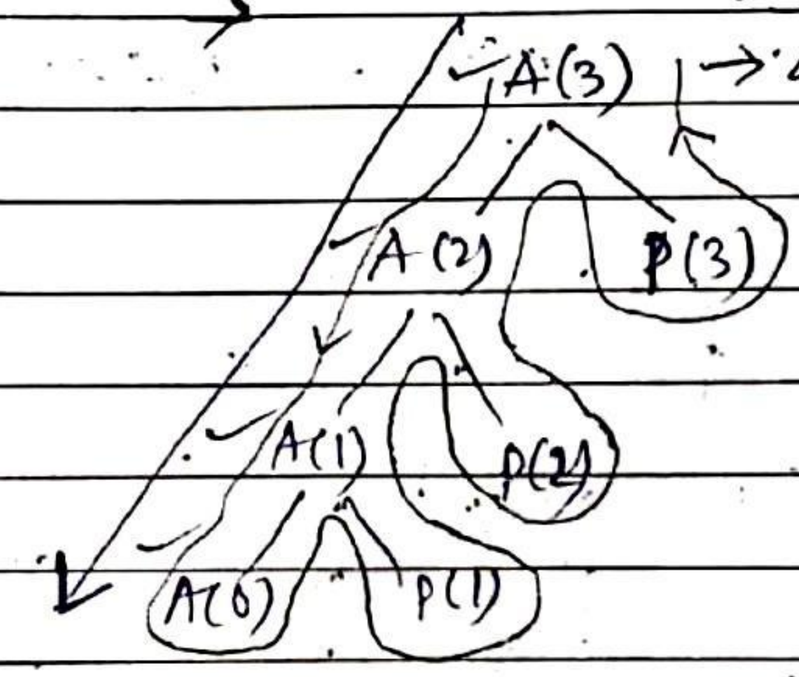
• Space complexity of recursive (Algorithm) program =

→ When ~~the execution program~~ no. of statement less inside the program then use tree method.

ex-(1)

```
A(n)  — (n+1)
{ if(n>=1)
  {
    A(n-1);
  }
  Pf(n);
}
```

✓ Space complexity = O(n)

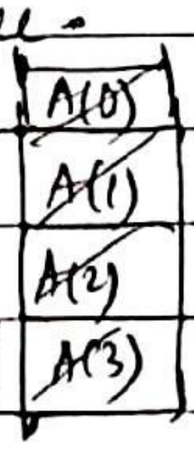


output = 1 2 3

for A(3) function called - 4 times.

for A(n) function called - n times.

→ Space complexity is depth of the tree.



n - (n+1)K
- O(kn)
- O(n)

Time complexity =
recursive equation -

$$= 0 \quad n=0$$

$$= 1 \quad n=0$$

$$T(n) = T(n-1) + 1 \quad ; n \geq 1$$

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

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

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

$$\vdots$$

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

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

$$= T(0) + n$$

$$= 1 + n$$

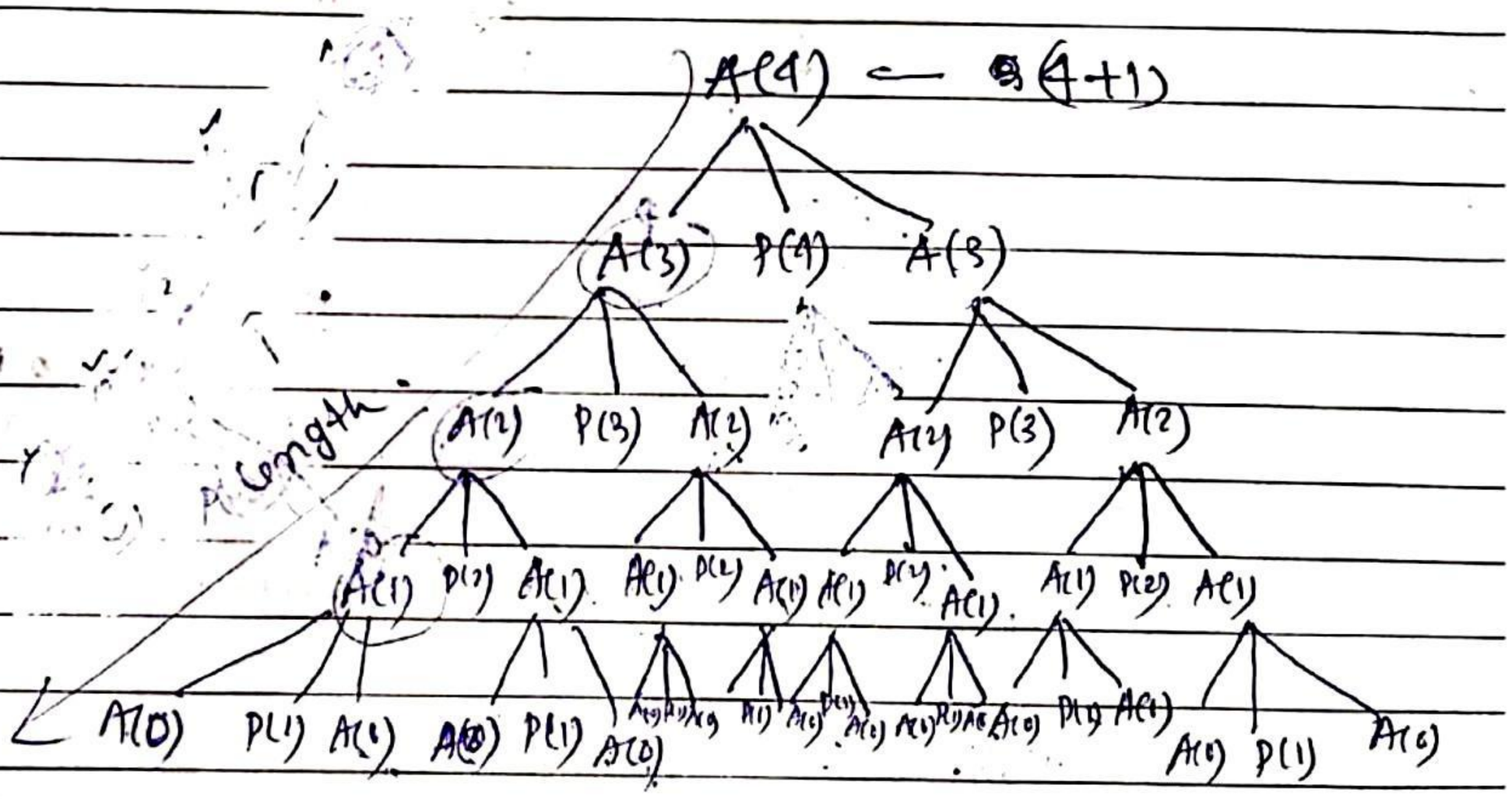
$$T(n) = 1 + n$$

$$= O(n).$$

ex-② (Find Time & space Complexity)

```
A(n) ← (n+1)
{
  if (n>=1)
  {
1. A(n-1);
2. Pf(n);
3. A(n-1);
  }
}
```

→ n = 4 A(4)



here Space Complexity = O(n+1)

= O(n).

output = 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

A(4) = 31 = $2^{4+1}-1$

A(3) = 15 (times function call) = $2^{3+1}-1$

A(2) = 7 = $2^{2+1}-1$

A(1) = 3 = $2^{1+1}-1$

⋮

A(n) = $2^{n+1}-1$ (for n value function called)

Time complexity =

Recursive equation =

$$T(n) = 2T(n-1) + 1 + T(n-1)$$
$$= 2T(n-1) + 1 \quad ; n \geq 1$$
$$= 1 \quad ; n = 0$$

$$T(n) = 2T(n-1) + 1 \quad \text{—(i)}$$
$$T(n-1) = 2T(n-2) + 1 \quad \text{—(ii)}$$
$$T(n-2) = 2T(n-3) + 1 \quad \text{—(iii)}$$

$$T(n) = 2(2T(n-2)+1) + 1$$
$$= 2 \cdot 2T(n-2) + 2 + 1$$
$$= 2^2(2T(n-3)+1) + 2 + 1$$
$$= 2^3 T(n-3) + 2^2 + 2^1 + 2^0$$
$$\vdots$$
$$= 2^K T(n-K) + 2^{K-1} + 2^{K-2} + \cdots + 1$$

$$= 2^n T(0) + 2^{n-1} + 2^{n-2} + \cdots + 1 \qquad \boxed{\begin{matrix} n-K=0 \\ K=n \end{matrix}}$$
$$= 2^n \cdot 1 + 2^{n-1} + 2^{n-2} + \cdots + 1$$
$$= 2^n + 2^{n-1} + 2^{n-2} + \cdots + 2^0 \quad \text{— GP}$$

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

$$= 2^{n+1} - 1$$
$$= O(2^{n+1})$$

$T(n) = O(2^n)$. (Time complexity)