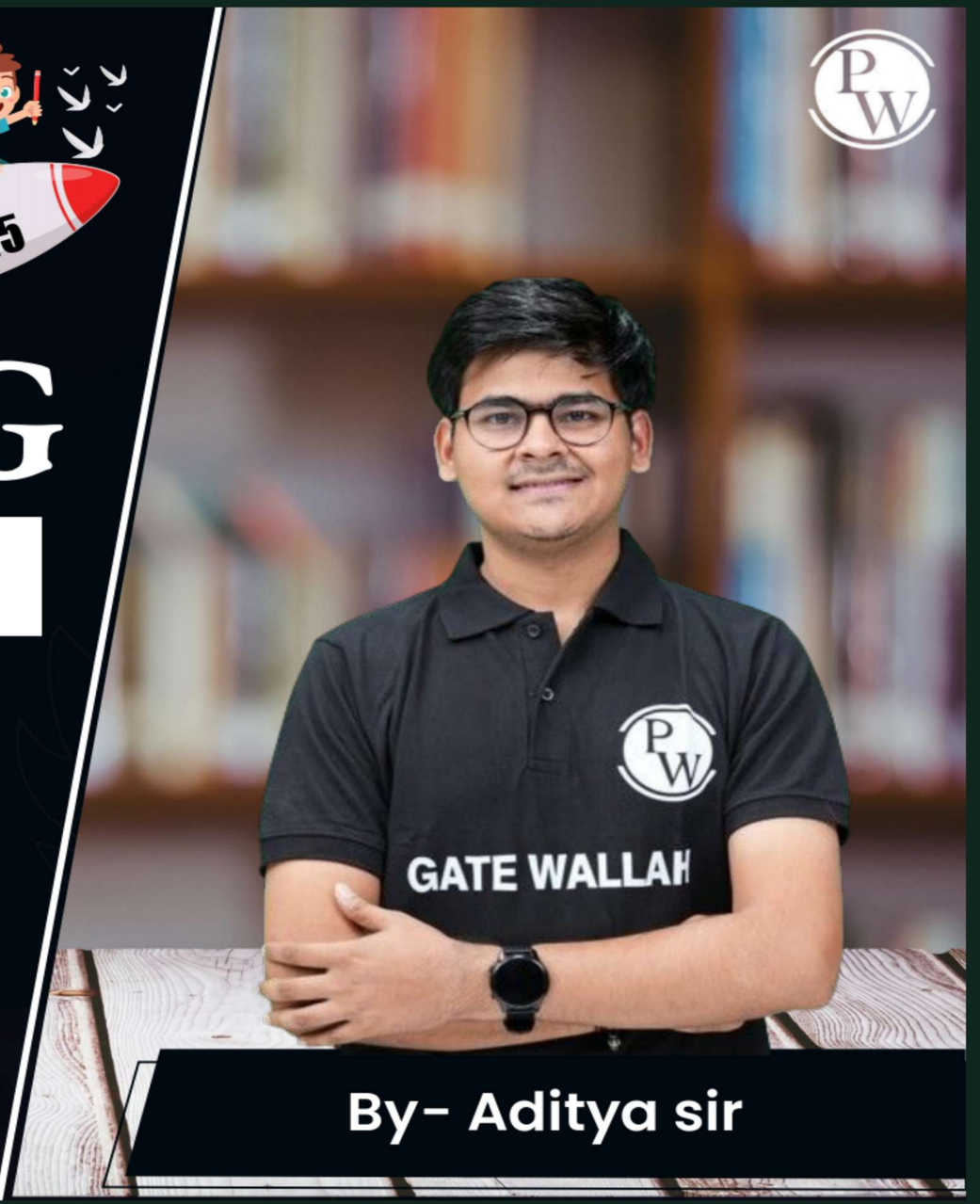
CS & IT ENGINEERING

Algorithms

Analysis of Algorithms

Lecture No.- 08



By- Aditya sir

Recap of Previous Lecture

Topic Problem Solving with ASNs * (PYQ / Practice)

Topic

Topic

Topics to be Covered

- **Topic** Framework for Analysing Recursive Algo
- **Topic** Problem Solving - Practice + PYQ
- **Topic**

Previous Year Questions

Recursion :-

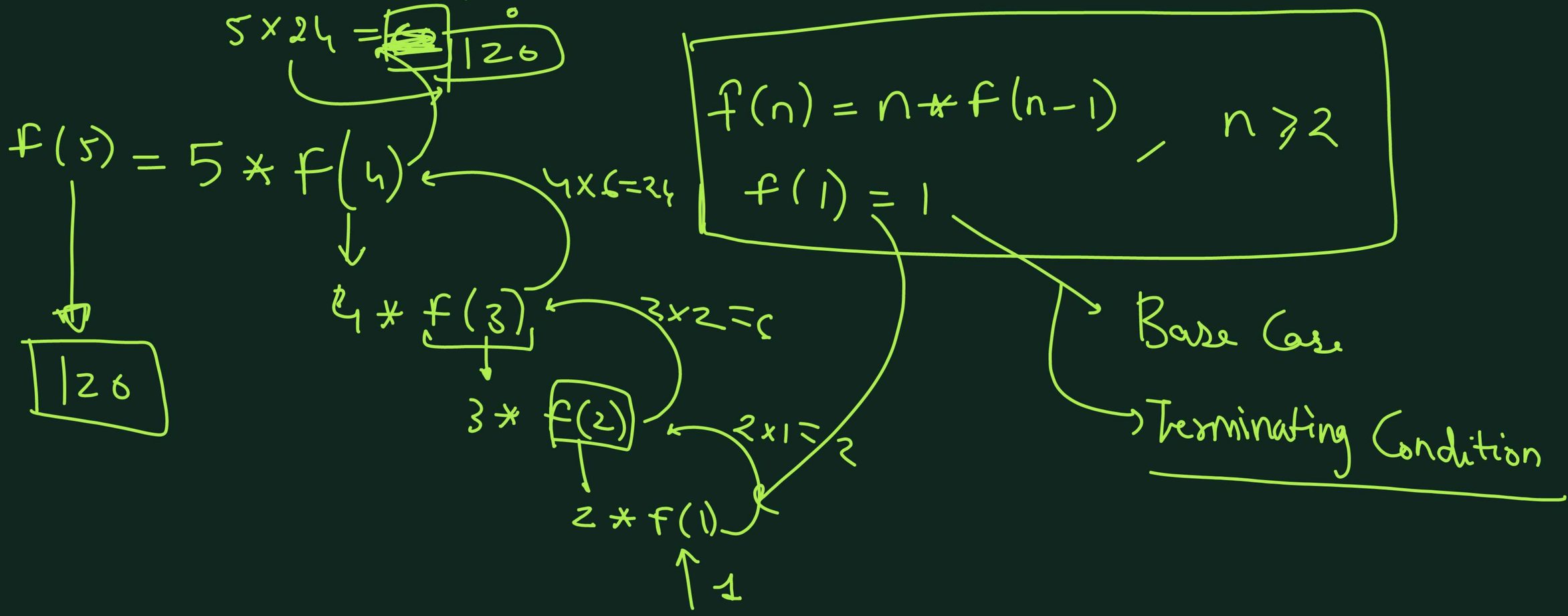
$n! = n \times n-1 \times n-2 \cdots \times 1$

$f(n) = n * f(n-1)$, $n \geq 2$

$f(1) = 1$

Base Case

Terminating Condition



Topic : Asymptotic Notations

$f(n) = \log_2 n$? $g(n) = \log_{10} n$

App 2

Appl : Substitute a value.

$\log_2 8$

$= \log_2(2^3)$

$= 3 * \log_2(2)$

$= 3 * 1$

$n = 10$

$f(n) = \log_2(10)$

$= 3.32$

$f > g$

$g(n) = \log_{10}(10)$

$= 1$

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

$\log_2 n$

$\frac{\log_{10} n}{\log_{10} 2}$

$\log_{10} n$

$\log_{10} n$

$\log_4 2 > 1$

$\log_{10}^{2} = 0.3$

$\frac{1}{\log_{10}^{2}} = \frac{1}{0.3} = 3.3 > 1$

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

App 3 :- $\log_2 n$ $\log_{10} n$

$\cancel{\log_2 n}$

$\frac{\cancel{\log_2 n}}{\log_2 10}$

$\frac{1}{\log_2 10}$

1

$\frac{1}{1} > \frac{1}{3.3}$

Topic : Asymptotic Notations

1. The time complexity of recursive algorithm is derived/represented as a recursive equation;
2. Solve recurrence to get the value of recurrence as a mathematical function (poly, Log, exp, constant)
3. Apply = O, Ω, θ

Framework

① Recursive Eqn (Recurrence)

② Solve above to get a Mathematical funct.

③ Apply ASN on above

Topic : Asymptotic Notations

Solving Recurrences

1. Back Substitution
   - Repeated Substitution
   - (general approach)
2. Master Method
   - (shortcut Appr)
3. Recursion Tree

Topic : Time Complexity Framework for Recursive Algorithms

```
1.  Algorithm what (n)
    {
        if (n = 1) return;
        else
        {
            what (n – 1);
            B(n);
        }
    }
```

→ T(n)

→ if (n = 1) return; → (a)

Recursion

external function call

what (n – 1); → T(n–1)

B(n); → (b) → B(n) → n (Case 2)

(i) Assum B(n) = O(1)

Let T(n) represent the time Complexity of the Algo what(n).

↳ GATE + Placement

Case 3 :- B(n) = O(1/n)

$$T(n) = c \quad , \quad n=1 \quad , \quad c>0$$

$$T(n) = a + T(n-1) + b \quad , \quad n>1 \qquad [\, a \geq 0 \, , \, b \geq 0 \,]$$

$$= T(n-1) + d \quad \text{---} (1) \qquad [\, d = a+b \,] \geq 0$$

$$\boxed{T(n) = T(n-1) + d} \rightarrow \text{Recurrence}$$

Step 2

Back Substitution:

$T(n) = c, \quad n = 1$

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

$T(n) = T(n-1) + d \quad \text{---} (1)$

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

$= T(n-2) + d \quad \text{---} (2)$

using (2) in (1)

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

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

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

$T(n) = T(n-3) + 3d \quad \text{---} (3)$

generalised form

$T(n) = T(n-k) + k * d \quad \text{---} (4)$

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

for base condition,

$(n-k) = 1$

$k = (n-1)$

$T(n) = T(n-k) + k * d$.

$k = (n-1)$

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

$T(n) = C + (n-1) * d$

(Mathematical eqn)

Soln / value of the Recurrance Relation

Step 3 :-

$O(T(n)) \Longrightarrow O(n)$, $\Omega(n)$ $\therefore \Theta(n)$

Quiz: Assume, $B(n) = O(n)$

$$T(n) = c, \quad n=1$$

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

$$T(n) = a + T(n-1) + n \quad \text{---} \quad (1)$$

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

$$T(n) = a + \left(a + T(n-2) + (n-1) \right) + n$$

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

$$= 2a + \left(a + T(n-3) + (n-2) \right)$$

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

Generalised eqn

$$T(n) = k*a + T(n-k) + n + (n-1) + (n-2) \cdots + (n-(k-1))$$

$(n-k) = 1$, $k = (n-1)$

$$T(n) = (n-1)*a + T(1) + n + (n-1) + \cdots + n-(n-1-1)$$

$$= (n-1)*a + T(1) + \underbrace{n + (n-1) + (n-2) \cdots 2}$$

$(2+3+\cdots n)$

$$T(n) = (n-1)*a + T(1) + \left(\sum_{1}^{n} n - 1\right)$$

$$= (n-1)*a + \underbrace{T(1)}_{c} + \frac{n(n+1)}{2} - 1$$

$$= an - a + c + \frac{n^2+n}{2} - 1$$

Step2 →

$$T(n) = \left[\frac{n^2}{2} + \left(\frac{1}{2}+a\right)n + A'\right]$$

$$A' = c - a - 1$$

Step3: $O(n^2)$ / $\Omega(n^2)$ $\therefore \Theta(n^2)$

Case 3 → PYQ : (2 mks) Assume $B(n) = O(1/n)$

$$T(n) = c, \quad n = 1$$

$$T(n) = a + T(n-1) + \frac{1}{n}, \quad n > 1 \quad \text{— (1)}$$

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

$$T(n) = a + \left(a + T(n-2) + \frac{1}{(n-1)} \right) + \frac{1}{n}$$

$$= 2a + T(n-2) + \frac{1}{n} + \frac{1}{n-1} \longrightarrow (3)$$

generalised form

$$T(n) = k*a + T(n-k) + \frac{1}{n} + \frac{1}{n-1} \cdots + \frac{1}{n-(k-1)}$$

$$n - k = 1$$

$$k = (n-1)$$

$$T(n) = (n-1)*a + T(1) + \left[\frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} \cdots + \frac{1}{2} \right]$$

$$\sum_{x=1}^{n} \frac{1}{x} = \log(n)$$

$$T(n) = (n-1) * a + \underbrace{T(1)}_{c} + \log n - 1$$

$$= a(n-1) + c + \log n - 1$$

$$= an - a + c + \log n - 1$$

Step 2:

$$T(n) = an + \log n + \underbrace{(c - a - 1)}_{\text{const}}$$

Step 3: $O(n)$ ✓

$$\sum_{n=1}^{n} \frac{1}{n} = \log n$$

Topic : Time Complexity Framework for Recursive Algorithms

2. Algorithm Do_it(n)
{
if (n = 1) return;
else
return (Do_it (n–1) + Do_it(n–1));
}

Ans

Expo → O(2^n) ①

a

T(n–1) + T(n–1)

b

$T(n) = c$, $n=1$, $c>0$

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

$= 2T(n-1) + d$, $(d = a+b) > 0$

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

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

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

$$= 2^2 T(n-2) + 3d \quad \text{---} (2)$$

$(4-1)$

$$= 2^3 T(n-3) + 7d \quad \text{---} (3)$$

$(8-1)$

generalised

$$T(n) = 2^k T(n-k) + (2^k - 1) * d$$

$$n - k = 1 \longrightarrow \boxed{k = (n-1)}$$

$$T(n) = 2^{(n-1)} * T(1) + (2^{(n-1)} - 1) * d$$

$T(1) \downarrow c$

$$= 2^{(n-1)} * c + 2^{(n-1)} * d - d$$

$$= 2^{(n-1)} * (c+d) - d$$

$$= 2^n * \left(\frac{c+d}{2}\right) - d$$

$$\boxed{T(n) = 2^n * e - d}$$

$$O(2^n)$$ ✓

Topic : Time Complexity Framework for Recursive Algorithms

→ PYQ (2m)

3. Algorithm A(n)

```
{
    if (n = 2) return;        → (a)
    else
    return (A(√n));
                  → T(√n)
}
```

Step1 : $T(n) = c$, $n=2$

$T(n) = a + T(\sqrt{n})$, $n>2$

$T(n) = a + T(\sqrt{n}) \rightarrow (1)$

$T(\sqrt{n}) = a + T(n^{1/4})$

$\sqrt{n} = n^{1/2}$

$(\sqrt{n})^{1/2} = n^{(1/4)}$

$T(n) = a + (a + T(n^{1/4}))$

$= 2a + T(n^{1/4}) \rightarrow (2)$

$= 3a + T(n^{1/8}) \rightarrow (3)$

Generalised,

$n^{1/2}$, $n^{1/4}$, $n^{1/8}$ ⟶ $n^{1/2^k}$

$n^{1/2^1}$, $n^{1/2^2}$, $n^{1/2^3}$

$$T(n) = T\left(n^{1/2^k}\right) + k*d \longrightarrow (3)$$

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

$$\log_2\left(n^{1/2^k}\right) = \log_2 2$$

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

$$2^k = \log_2 n$$

$$2^k = \log_2 n$$

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

$$k * \log_2 2 = \log(\log n)$$

$$k = \log(\log n)$$

$$T(n) = T(2) + k*d$$

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

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

$$O(\log(\log n))$$

Topic : Time Complexity Framework for Recursive Algorithms

4. Algorithm A(n)

```
{
    if (n = 2) return;
    else
    return ((A(√n)) + (A(√n)));
}
```

Sol:-

$T(n) = c, \quad n = 2$

$T(n) = a + T(\sqrt{n}) + T(\sqrt{n}) + b$

$T(n) = 2T(\sqrt{n}) + d, \quad (d = a+b) > 0.$ → (1)

$T(\sqrt{n}) = 2T(n^{1/4}) + d$

$T(n) = 2(2T(n^{1/4}) + d) + d$

$= 2^2 T(n^{1/4}) + 3d$ → (2)

$= 2^3 T(n^{1/8}) + 7d$ → (3)

generalised eqn

$$T(n) = 2^k * T\left(n^{1/2^k}\right) + \left(2^k - 1\right) * d \longrightarrow (5)$$

BC: $n^{1/2^k} = 2$, $2^k = \log(n)$

$k = \log\log n$

$$T(n) = \log n * \underbrace{T(2)}_{c} + (\log n - 1) * d$$

$T(n) = c * \log n + d * \log n - d$

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

$3/4 = 1 - 1/2^2$

Topic : Time Complexity Framework for Recursive Algorithms

5. $T(n) = 2$, $n = 2$
   $= \sqrt{n}.T(\sqrt{n}) + (n)$, $n > 2$

$T(n) = 2$, $n = 2$

Type 2

$n^{1/2} * n^{1/4} = n^{\frac{1}{2}+\frac{1}{4}} = (n^{3/4})$

$T(n) = \sqrt{n} * T(\sqrt{n}) + n$ — ①

$T(\sqrt{n}) = n^{1/4} * T(n^{1/4}) + \sqrt{n}$

$T(n) = n^{1/2}\left[n^{1/4} * T(n^{1/4}) + n^{1/2}\right] + n^1$

$T(n) = n^{3/4} * T(n^{1/4}) + n + n$

$= n^{1-\frac{1}{2^2}} * T(n^{1/2^2}) + 2n$ — ②

$= n^{1-\frac{1}{2^3}} * T(n^{1/2^3}) + 3n$ — ③

$= n^{\left(1-\frac{1}{2^k}\right)} * T(n^{1/2^k}) + kn$ — ④

$$n^{a-b} = \frac{n^a}{n^b}$$

generalised

$$T(n) = n^{\left(1-\frac{1}{2^k}\right)} * T\left(n^{1/2^k}\right) + k * n \rightarrow (4)$$

B.C : $n^{1/2^k} = 2$, $2^k = \log n$

$k = \log(\log n)$

$$T(n) = \left(\frac{n^1}{n^{1/2^k}}\right) * T(2) + \log(\log n) * n$$

$$= \frac{n}{2} * 2 + (\log \log n) * n$$

$$= n + (\log \log n)\, n$$

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

Topic : Time Complexity Framework for Recursive Algorithms

6. Algorithm Recur(n)

```
{
    if(n = 1) return;
    else
    {
        Recur(n/2);
        Recur(n/2);
        B(n);
    }
}
```

Case 1 :- Assume : $B(n) = O(1)$; Case 2 : $B(n) = O(n)$

if(n = 1) return; → (a)

Recur(n/2); → $T(n/2)$

Recur(n/2); → $T(n/2)$

B(n); → $O(1)$ → b

$$T(n) = c, \quad n = 1$$

$$T(n) = a + T(n/2) + T(n/2) + b$$

$$= 2T(n/2) + d, \quad (d = a + b) > 0$$

$$T(n) = 2T(n/2) + d \quad \text{---} (1)$$

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

$$T(n) = 2\left[2 * T(n/2^2) + d\right] + d$$

$$= 2^2 * T\left(\frac{n}{2^2}\right) + \overbrace{3d}^{(2^2-1)} \quad \text{---} (2)$$

$$= 2^3 * T\left(\frac{n}{2^3}\right) + \overbrace{7d}^{(2^3-1)} \quad \text{---} (4)$$

generalised

$$T(n) = 2^k * T\left(\frac{n}{2^k}\right) + (2^k - 1)d$$

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

$$T(n) = n * \underbrace{T(1)}_{c} + (n-1) * d$$

$$= n * c + n * d - d$$

$$T(n) = O(n)$$

Case 2 :- $B(n) = O(n)$

$$T(n) = c, \quad n = 1$$

$$T(n) = 2T(n/2) + a + n, \quad n > 1$$

$$T(n/2) = 2T(n/4) + a + n/2 \quad (\star)$$

$$T(n) = 2\left(2 * T\left(\frac{n}{2^2}\right) + a + \frac{n}{2}\right) + a + n$$

$$T(n) = 2^2 * T\left(\frac{n}{2^2}\right) + 2a + n + a + n$$

$$= 2^2 * T\left(\frac{n}{2^2}\right) + 3a + 2n \quad \text{---(2)}$$

$$= 2^3 * T\left(\frac{n}{2^3}\right) + 7a + 3n \quad \text{---(3)}$$

generalised

$$T(n) = 2^k * T\left(\frac{n}{2^k}\right) + (2^k - 1)a + k * n \longrightarrow (4)$$

BC

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

$2^k = n$

$k = (\log n)$

$$= n * T(1) + (n-1) * a + (\log n)\, n$$

$T(1) = c$

$$= cn + an - a + n \log n$$

Ans

$$O(n \log n)$$

Topic : Time Complexity Framework for Recursive Algorithms

7. The given diagram represents the flowchart of recursive algorithm A(n). Assume that all statement except for the recursive calls have order (1) time complexity. Then the best case and worst case time of this algorithm is _____.

O( ) O( )

?

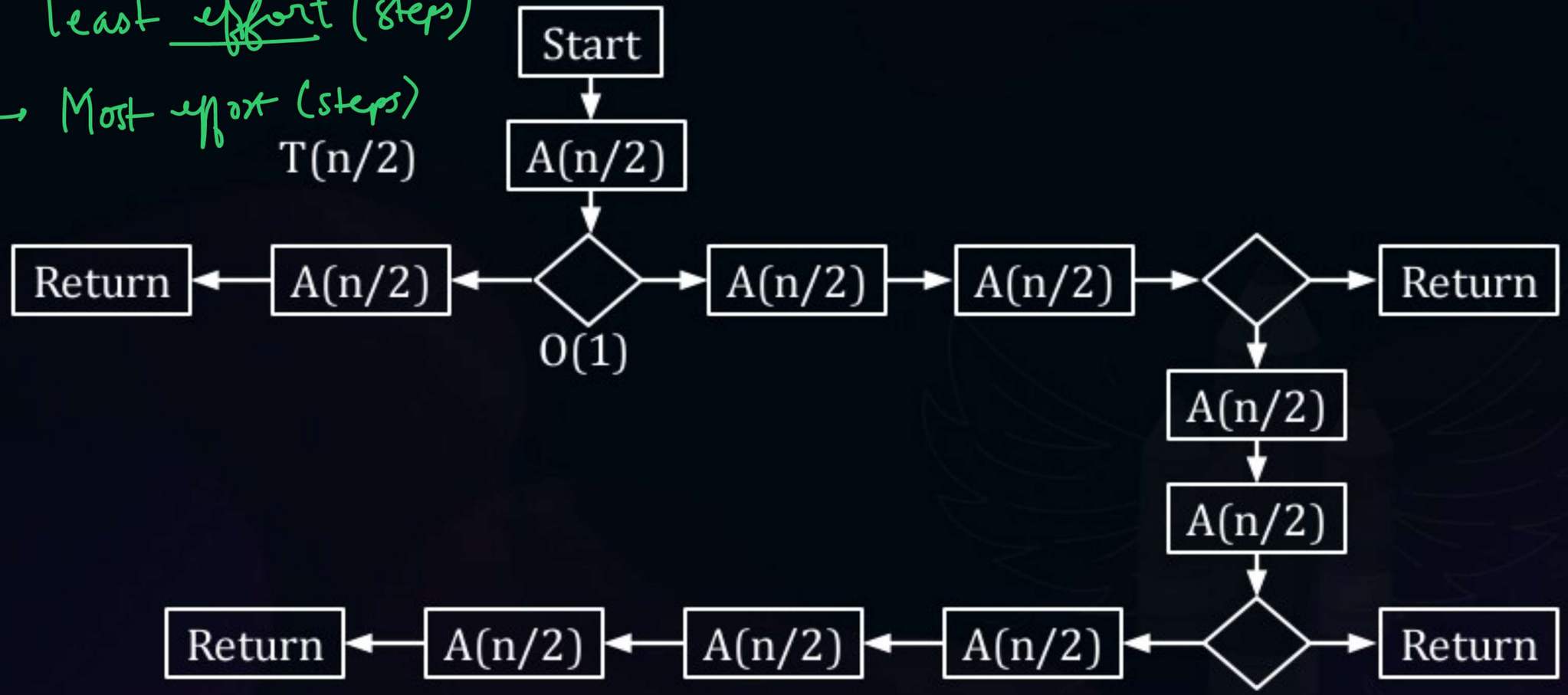
→ [1+W1]

(diag on next page)

Topic : Time Complexity Framework for Recursive algorithms

BC ⟶ least effort (steps) Algo

WC ⟶ Most effort (steps)



Hw2 :-

(i) $B(n) \longrightarrow O(1)$

(ii) $B(n) \longrightarrow O(n)$

```
Algo Do-it(n)
{
    if (n==1) return;
    else
    {
        Do-it(n/2)
        B(n);
    }
}
```

HW3:-

$$T(n) = 2T(n/2) + n \log n, \quad n > 1$$

$$= c, \quad n = 1$$

2 mins Summary

Topic Framework for Analyzing Recursive Algorithms

Topic Problem Solving - Practice + PYQ

THANK - YOU