CS & IT ENGINEERING

Algorithms

Introduction to Algorithms and Analysis

Lecture No.- 08

By- Dr. Khaleel Khan Sir

Recap of Previous Lecture

Topic Problem Solving with ASNs

Topic

Topic

Topic

Topic

Topics to be Covered

Topics

- Framework for Analysing Recursive Algo
- Framework for Analyzing Non- Recursive Algo

Topic: Asymptotic Notations

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

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

(i) $n = 10$

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

$= 3.32 \; > \; = 1$

(ii) $\log_2 n \; > \; \log_{10} n$

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

$$\frac{1}{\log_{10} 2} \qquad 1$$

$$\left(\frac{1}{0.3}\right) = 3.33 \; > \; 1$$

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

$$\boxed{g(n) \text{ is } O(f(n))}$$

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

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

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

1. → The Time Complexity of Recursive Algorithm is derived / represented as a Recurrence Equation;

   → Engg. Maths / D.M (Mathematical Model)

2. → Solve Recurrence to get the Value of a Recurrence as a Mathematical fn (Poly; Log; Expo; Constant)

3. → Apply: O, Ω, Θ

Solving recurrences

1) Back Substitution

⟨Universal Method⟩

Repeated Substitution

2) Master Method

⟨Short-Cut⟩

3) Recursion Tree

Topic : Time Complexity Framework for Recursive Algorithms

Devlop (Recursion)

1. Algorithm What (n)

```
{
    if (n = 1) return;
    else
    {
        What (n – 1);
        B(n);
    }
}
```

a

function

(i) Assume $B(n) = O(1)$

Let $T(n)$ repr. Time Complexity of What(n)

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

$\boxed{T(n) = a + T(n-1) + b} \quad , n > 1 \quad [a > 0 ; b > 0]$

→ Recurrence

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

PYQ: 2M

(ii) Assume $B(n) = O(n)$

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

$\quad = a + T(n-1) + n \quad — ①$

(iii) Assume $B(n) = O(1/n)$

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

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

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

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

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

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

$$= T(n-3) + 3d \quad - (4)$$

Generalization $= T(n-K) + K \cdot d \quad - (5)$

$$= T(1) + K \cdot d$$

$$= C + K \cdot d$$

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

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

$$n - K = 1$$

$$K = n - 1$$

$$\boxed{T(n) = C + dn - d}, \quad n > 1$$

Soln / value of recurrence

$O(n)$

$\Omega(n)$

$\Theta(n)$ ✓

(ii) $T(n) = T(n-1) + a + n$ — ①

$T(n-1) = T(n-2) + a + (n-1)$ — ②

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

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

$$= T(n-3) + (n-2) + (n-1) + n + 3a \quad — ④$$

$$\vdots$$

$$= T(n-k) + (n-(k-1)) + \cdots + (n-1) + n + k \cdot a \quad — ⑤$$

$$= T(1) + (\cancel{n} - (\cancel{n}-1-1)) + \cdots + n + k \cdot a$$

$$= c + \underbrace{(2+3+4 \cdots + n)}_{\frac{n(n+1)}{2} - 1} + a(n-1)$$

$n-k=1$

$k=(n-1)$

Lt $c=1$

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

$$= \sum_{i=1}^{n} i + an - a$$

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

$\to O(n^2)$

$\Omega(n^2)$

$\therefore \Theta(n^2)$

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

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

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

$$= T(n-k) + \frac{1}{(n-(k-1))} + \cdots + \frac{1}{n} + k \cdot a \quad - (4)$$

$$= T(1) + \left(\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}\right) + a(n-1) \quad - (5)$$

$$= c + \quad " \quad + an - a$$

Let $c = 1$

$$n - k = 1$$

$$\Rightarrow k = n - 1$$

$$T(n) = \sum_{x=1}^{n} 1/x + an - a$$

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

$$O(n)$$

$$\Omega(n) \quad \therefore \Theta(n)$$

$$T(n) = \begin{cases} T(n-1) + \dfrac{1}{n}, & n > 1 \\ c, & n = 1 \end{cases} \Rightarrow O(\log n)$$

PYQ

Topic : Time Complexity Framework for Recursive Algorithms

2. Algorithm Do_It (n) : Expo

$O(2^n)$

```
{
    if (n = 1) return;
    else
    return (Do_It (n – 1) + Do_It (n – 1));
}
```

$T(n-1)$ $T(n-1)$

$\hookrightarrow V_1$ V_2

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

$= a + 2 \cdot T(n-1) + b$

$T(n) = 2 \cdot T(n-1) + d \quad (d = a+b)$

$T(n) = 2T(n-1) + d$ — (1) — $2^1 \cdot T(n-1) + (2^1 - 1) \cdot d$

$T(n-1) = 2T(n-2) + d$ — (2)

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

$= 4T(n-2) + 3d$ — (3)

$= 2^2 \cdot T(n-2) + (2^2 - 1) \cdot d$

$= 2^3 \cdot T(n-3) + (2^3 - 1) \cdot d$

$= 2^K \cdot T(n-K) + (2^K - 1) \cdot d$ — (4)

$= 2^K \cdot T(1) + 2^K \cdot d - d$

$= 2^{n-1} \cdot c + 2^{n-1} \cdot d - d$

$n - K = 1$

$K = n - 1$

$T(n) = e \cdot 2^n - d$

$O(2^n)$

$\Theta(2^n)$

Slide 6

$$f(n) = n * f(n-1)$$
$$f(1) = 1$$

$$f(5) = 5 * f(4)$$
$$= 5 * [4 * f(3)]$$
$$= 5 * [4 * 3 * f(2)]$$

$$f(1)$$
$$1$$

PYQ : 2M

3. Algorithm A(n)

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

$n>2$

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

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

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

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

$T(n) = T(n^{1/4}) + 2a \quad - (3)$

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

$= T(n^{1/2^3}) + 3a$

$= T(n^{1/2^K}) + K \cdot a$

$= T(2) + a \cdot K$

$= C + a \cdot \log\log n$

$T(n)$ in $O(\log\log n)$

$\boxed{n^{\frac{1}{2^K}} = 2}$

$\frac{1}{2^K} \cdot \log n = \log_2 2$

$2^K = \log n$

$\boxed{K = \log\log n}$

PYQ : 2M

Topic : Time Complexity Framework for Recursive Algorithms

4. Algorithm A(n)

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

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

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

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

$T(n) = 2 \cdot T(n^{1/2}) + d$ — 1

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

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

$= 4 \cdot T(n^{1/4}) + 3d$ — 3

$= 2^2 \cdot T(n^{1/2^2}) + (2^2 - 1) \cdot d$

$= 2^3 \cdot T(n^{1/2^3}) + (2^3 - 1) \cdot d$

$\vdots$

$= 2^K \cdot T(n^{1/2^K}) + (2^K - 1)\, d$

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

$\Rightarrow 2^K = \log n$

$T(n) = 2^K \cdot T(n^{1/2^K}) + (2^K - 1) \cdot d$

$= \log n \cdot c + \log n \cdot d - d$

$O(\log n)$

$2^{\log_2 \log_2 n} = \log n^{\log_2 2}$

Slide 8

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

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

$$T(n) = n^{1/2} \cdot T(n^{1/2}) + n \quad - (1)$$

$$T(n^{1/2}) = n^{1/4} \cdot T(n^{1/4}) + n^{1/2} \quad - (2)$$

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

$$= n^{3/4} \cdot T(n^{1/4}) + 2n \quad - (3)$$

$$\left(3/4 = 1 - \frac{1}{2^2}\right)$$

$$T(n) = n^{1-\frac{1}{2^2}} \cdot T(n^{1/2^2}) + 2n \quad - (4)$$

$$= n^{1-\frac{1}{2^3}} \cdot T(n^{1/2^3}) + 3n \quad - (5)$$

$$= n^{1-\frac{1}{2^k}} \cdot T(n^{1/2^k}) + k \cdot n \quad - (6)$$

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

$$k = \log\log n$$

$$= \frac{n}{n^{1/2^k}} \cdot T(2) + n \cdot k$$

$$= \frac{n}{\cancel{2}} \cdot \cancel{2} + n \cdot \log\log n = \boxed{n + n \cdot \log\log n}$$

$$T(n) = O(n \cdot \log\log n)$$

5. Algorithm Recur (n)

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

(i) Assume $B(n) = O(1)$

(ii) Assume $B(n) = O(n)$ *

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

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

$T(n) = 2T(n/2) + d \,, \quad — (1)$

$T(n/2) = 2T(n/4) + d \quad — (2)$

$T(n) = 2\left[2T(n/4) + d\right] + d$

$\quad = 4 \cdot T(n/4) + 3d \quad — (3)$

$\quad = 2^2 \cdot T(n/2^2) + (2^2 - 1) \cdot d$

$\quad = 2^3 \cdot T(n/2^3) + (2^3 - 1) \cdot d$

$T(n) = 2^K \cdot T(n/2^K) + (2^K - 1) \cdot d$

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

$\boxed{= cn + dn - d}$

$\rightarrow O(n)$ ✓

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

$\Rightarrow n = 2^K$

$$T(n) = 2 \cdot T(n/2) + n + a \quad - (1)$$

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

$$T(n) = 2\left[2T(n/4) + n/2 + a\right] + n + a$$

$$= 4 \cdot T(n/4) + 2n + 3a \quad - (3)$$

$$= 2^2 \cdot T(n/2^2) + 2n + (2^2 - 1) \cdot a \quad - (4)$$

$$= 2^3 \cdot T(n/2^3) + 3n + (2^3 - 1) a \quad - (5)$$

$$\vdots$$

$$= 2^K \cdot T(n/2^K) + K \cdot n + (2^K - 1) \cdot a \quad - (6)$$

$$= \boxed{n \cdot c + n \cdot \log n + an - a}$$

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

$$\Rightarrow n = 2^K$$

$$\Rightarrow K = \log n$$

$$\rightarrow O(n \cdot \log n)$$

$$\Omega(n \cdot \log n)$$

$$\Theta(n \cdot \log n)$$

Algo Do_it(n)
{
if (n==1) return;
else
{
Do_it(n/2);

B(n);
}
}

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

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

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

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

$T(n/2) = T(n/4) + d \quad - (2)$

$T(n) = T(n/4) + 2d \quad - (3)$

$\quad = T(n/2^2) + 2d \quad - (4)$

$\vdots$

$\quad = T(n/2^K) + K \cdot d \quad - (5)$

$\quad = T(1) + d \cdot \log n$

$\quad = c + d \cdot \log n$

$\quad = O(\log n)$ ✓

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

$\Rightarrow K = \log n$

(ii) $B(n) = O(n)$ *

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

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

$$T(n) = T(n/2) + n \quad - (1)$$
$$T(n/2) = T(n/4) + n/2 \quad - (2)$$
$$T(n) = T(n/4) + \frac{n}{2^1} + \frac{n}{2^0} \quad - (3)$$
$$= T\left(n/2^2\right) + \frac{n}{2^1} + \frac{n}{2^0} \quad - (3)$$
$$= T\left(n/2^3\right) + \frac{n}{2^2} + \frac{n}{2^1} + \frac{n}{2^0} \quad - (4)$$

$$T(n) = T(n/2^k) + \frac{n}{2^{k-1}} + \frac{n}{2^{k-2}} + \cdots + \frac{n}{2^1} + \frac{n}{2^0}$$
$$= T(1) + \frac{n}{2^0} + \frac{n}{2^1} + \frac{n}{2^2} + \cdots + \frac{n}{2^{k-1}}$$
$$= C + n\left[\sum_{i=0}^{k-1} 1/2^i\right]$$

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

$$= c + n\left[\frac{1\left(1 - 1/2^k\right)}{1 - 1/2}\right]$$

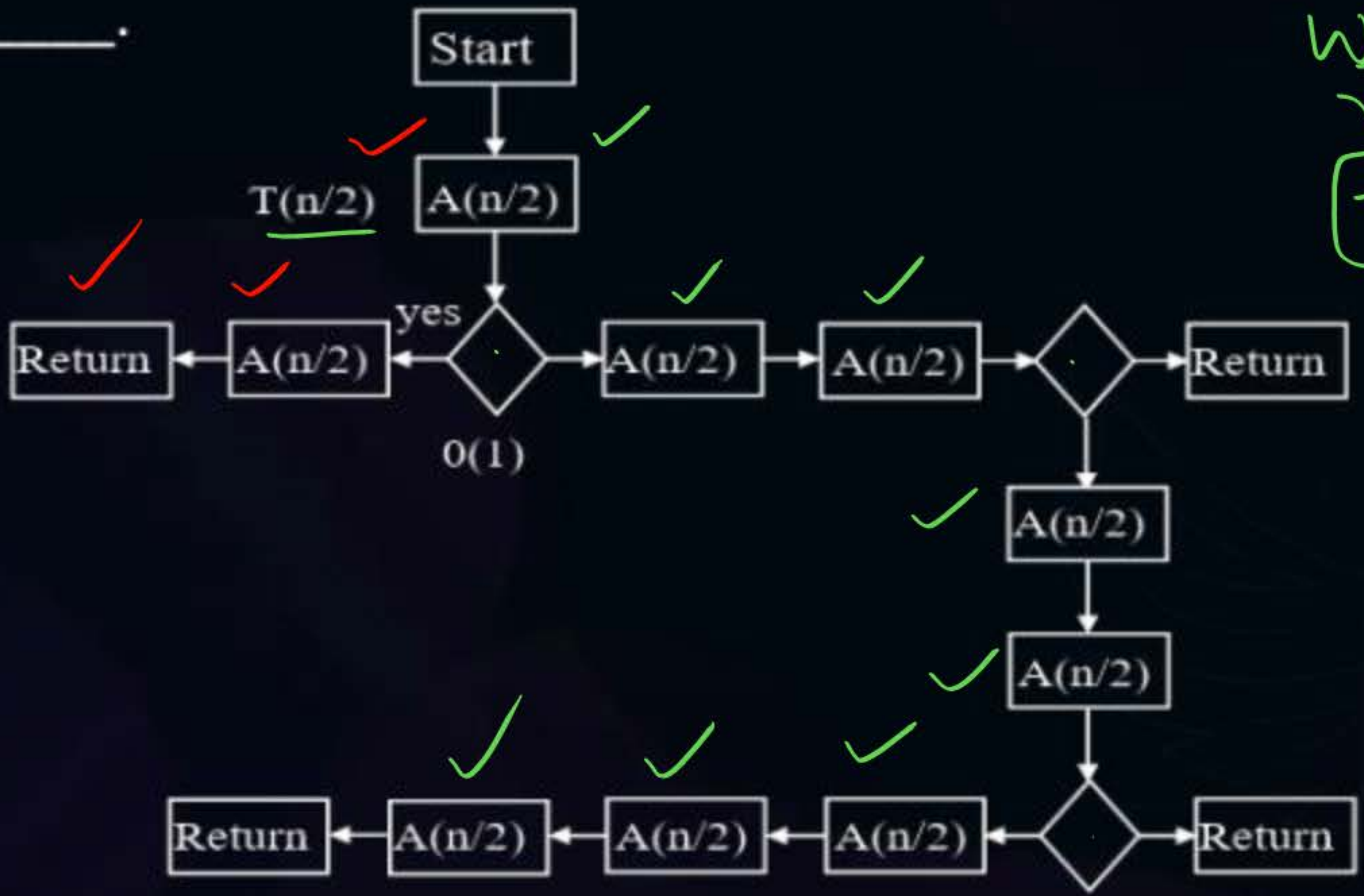
$$S_n = \frac{a(1 - r^n)}{1 - r}$$

$$= c + 2n\left[\left(1 - \frac{1}{2^k}\right)\right]$$
$$= c + 2n - \frac{2\cancel{n}}{\cancel{2^k}}$$

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

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

PYQ



Worst Case recurrence: O( )

$$\boxed{T(n) = 8 \cdot T(n/2) + c}$$

Best-Case:

$$\boxed{T(n) = 2 \cdot T(n/2) + c}$$

$\rightarrow O(n)$

Slide 11

$$T(n) = \begin{cases} \boxed{2 \cdot T(n/2) + n \cdot \log n} & , n > 1 \\ c & , n = 1 \end{cases}$$

H/w

THANK - YOU