CS & IT ENGINEERING

Algorithms

Analysis of Algorithms

Lecture No.- 11

By- Aditya sir

Recap of Previous Lecture

- Topic
- Topic
- Topic

Recursive Algo Time Complexity Analysis

Topics to be covered

Topic

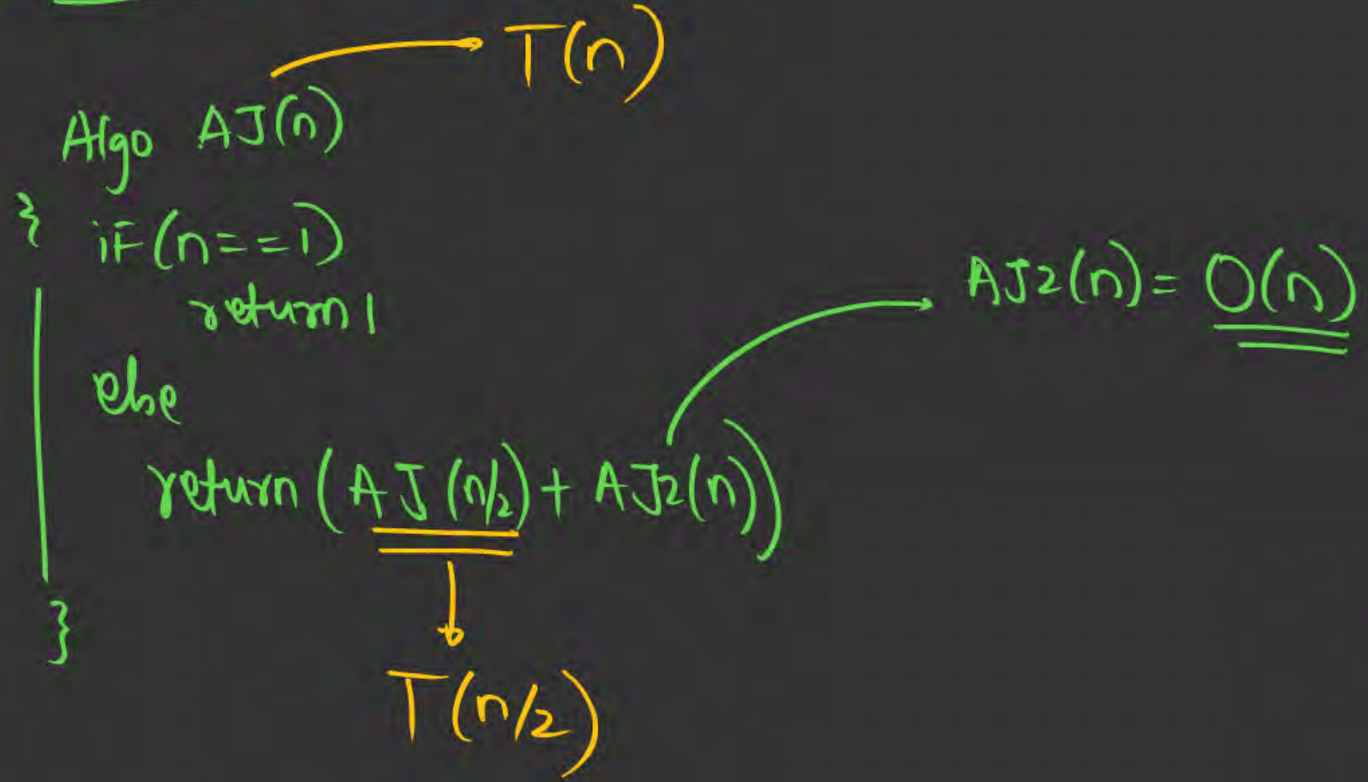
Topic

Topic

Time Complexity of Recursive Algo (new types)

Loop Complexity

Homework Soln:-



A) $O(n)$ ⟶ 39.5%

B) $O(\log n)$ ⟶ 15%

→ C) $O(n \log n)$ ⟶ 42%

D) $O(n^2)$ ⟶ 3.5%

Soln:

Step 1 : Recurrance relation.

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

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

Step 2 : Solve Recurrence using Back substitution.

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

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

$$T(n) = T(n/2^2) + n/2^1 + \frac{n}{2^0} + a + a$$

$$= T(n/2^2) + n/2^1 + n/2^0 + 2a$$

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

$$T(n) = T(n/2^3) + \left[n/2^2 + n/2^1 + n/2^0\right] + 3a \quad \text{— (2)}$$

$$T(n) = T(n/2^4) + \left[n/2^3 + n/2^2 + n/2^1 + n/2^0\right] + 4a \quad \text{— (3)}$$

⋮

Generalised eqn:-

$$T(n) = T(n/2^k) + \left[\frac{n}{2^{k-1}} + \frac{n}{2^{k-2}} \cdots \frac{n}{2^0}\right] + k * a \quad -③$$

$$n/2^{k-1} + n/2^{k-2} \cdots n/2^0 \Rightarrow$$

$$= n\left[\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} \cdots \frac{1}{2^0}\right]$$

$$= n\left[\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} \cdots \frac{1}{2^{k-1}}\right]$$

↳ GP: $a = 1$

$r = 1/2$

$n = (k)$

$$\Rightarrow \frac{a(1-r^n)}{1-r} = \frac{1 \times \left(1 - \frac{1}{2^k}\right)}{1 - 1/2}$$

$$= 2\left(1 - \frac{1}{2^k}\right)$$

$0 \longrightarrow k-1$

k terms

General term

$$T(n) = T\left(n/2^k\right) + n\left[2\left(1 - \frac{1}{2^k}\right)\right] + ka \quad \text{— (3)}$$

For Base Condition,

$$n/2^k = 1 \implies 2^k = n$$

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

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

$$= T(1) + n\left[2\left(1 - \frac{1}{n}\right)\right] + a * \log_2 n$$

$$T(n) = \left[b + 2n - 2 + a * \log_2 n\right]$$

Step 3 :- Apply asymptotic notation.

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

Topic : Time Complexity Framework for Recursive Algorithms

Imp

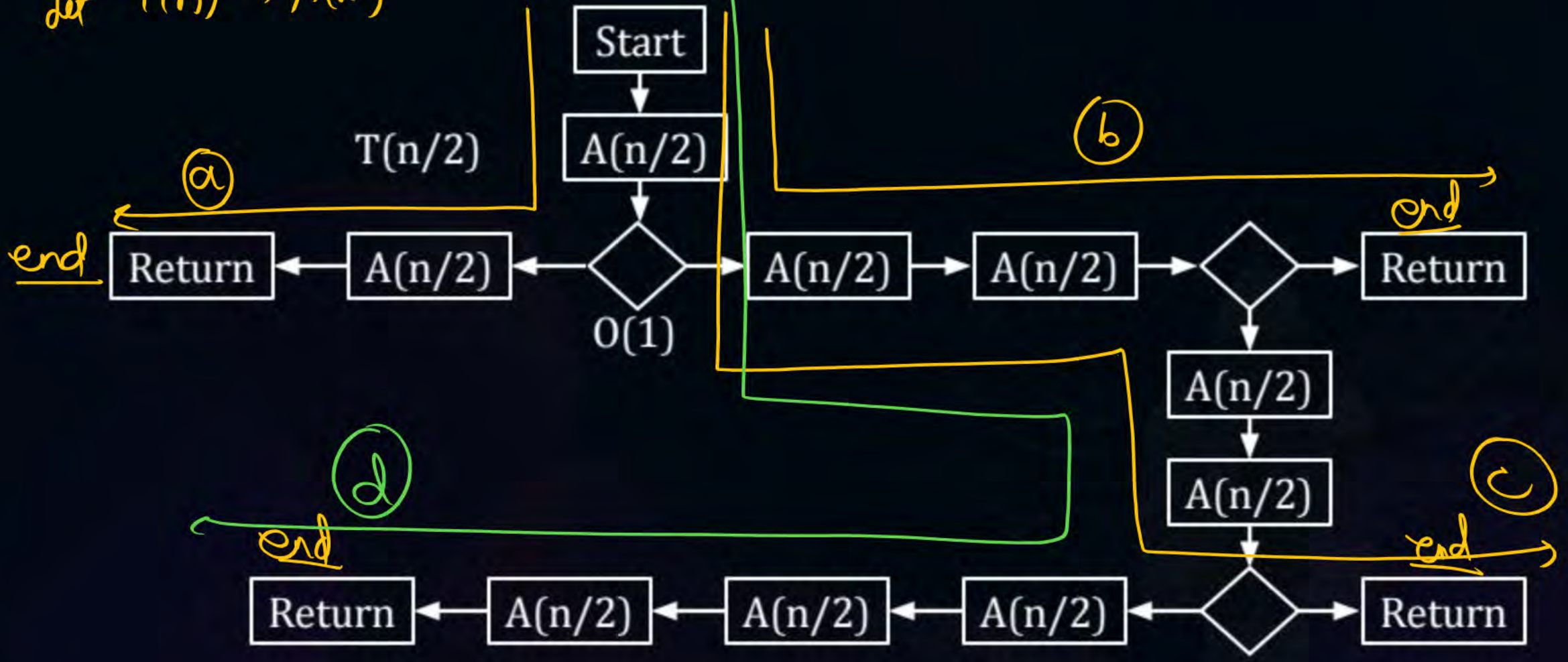
7. The given diagram represents the flowchart of recursive algorithm A(n). Assume that all statement except for the recursive calls have order (1) time complexity. Then the best case and worst case time of this algorithm is _____.

Best Case :- → Case when algo takes min steps/does min work/effort

Worst Case :- → Case when algo takes max steps/work/effort.

Topic : Time Complexity Framework for Recursive algorithms

Let $T(n) \rightarrow A(n)$



a) → (2) RC ⟶ Best Case

b) → 3 RC

c) → 5 RC

D) → 8 RC ⟶ Worst Case

76%

Case2

A) $O(n^2)$

~~B) $O(n^3)$~~

C) $O(n \log n)$

D) $O(n^2 \log n)$

Case1 :- Best Case Analysis:

Step1 :- Recurrance :

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

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

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

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

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

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

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

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

$T(n) = 2^k T(n/2^k) + (2^k - 1)a$.

for B.C,

$n/2^k = 1 \Rightarrow 2^k = n$.

$T(n) = n * T(1) + (n-1)a$.

$= n * b + an - a$

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

Case 2 : Worst Case

Step 1 : $T(n) = 8T(n/2) + a, \quad n > 1$ (the $8T(n/2)$ term: $A(n/2)$)

$= b, \quad n = 1$

Step 2: $T(n) = 8T(n/2) + a$

$T(n/2) = 8T(n/2^2) + a$

$T(n) = 8\left(8T(n/2^2) + a\right) + a$

$= 8^2 T(n/2^2) + 9a$ $(8+1)$

$T(n/2^2) = 8T(n/2^3) + a$

$T(n) = 8^2\left(8T(n/2^3) + a\right) + 9a$

$= 8^3 T(n/2^3) + 8^2 a + 9a$

$= 8^3 T(n/2^3) + \left(8^2 + 8^1 + 8^0\right)a$

General term.

$0 \longrightarrow k-1$

$T(n) = 8^k T(n/2^k) + \left(8^{k-1} + 8^{k-2} \cdots 8^0\right) a$.

GP: $a = 1$, $r = 8$, $n = k$. $\Rightarrow \frac{a(r^n - 1)}{r - 1} = \frac{(8^k - 1)}{8 - 1} = \frac{8^k - 1}{7}$

$$T(n) = 8^k T(n/2^k) + \left(\frac{8^k - 1}{7}\right) a \quad \text{— (1)}$$

For B.C, $n/2^k = 1$

$2^k = n$.

$8^k = (2^3)^k = (2^k)^3 = n^3$

$$T(n) = n^3 T(1) + \left(\frac{n^3 - 1}{7}\right) * a.$$

$\Rightarrow O(n^3)$

Given

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

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

↳ later, we will prefer master mtd.

Ans:- $O(n*(\log n)^2)$

A) $O(n\log n)$ → 38%

B) $O(n^2)$

C) $O(n$ 3%

D) none. — 3%

Step 1: $T(n) = 2T(n/2) + n\log n$ — ①

Step 2:- Back Substitution

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

$T(n/2) = 2T(n/2^2) + \frac{n}{2}\log(\frac{n}{2})$

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

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

$T(n/2^2) = 2T(n/2^3) + n/2^2 \log(n/2^2)$

$T(n) = 2^2\left[2T(n/2^3) + n/2^2 \log(n/2^2)\right] + n\log n + n\log n/2$

$= 2^3 T(n/2^3) + \left[n\log(n/2^2) + n\log n/2^0 + n\log(n/2^1)\right]$

$$T(n) = 2^3 T(n/2^3) + n\left[\log(n/2^0) + \log(n/2^1) + \log(n/2^2)\right]$$

$$\log a + \log b = \underline{\log(ab)}$$

$$T(n) = 2^3 T(n/2^3) + n\left[\log\left(n/2^0 \times n/2^1 * n/2^2\right)\right]$$

$$a^m \times a^n = a^{(m+n)}$$

Generalised term.

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

$$= 2^0 \times 2^1 \times 2^2 \cdots \times 2^{k-1}$$

$$= 2^{(0+1+2 \cdots k-1)}$$

$$0 \rightarrow k-1 \Rightarrow \left\{\frac{k(k-1)}{2}\right\}$$

$$T(n) = 2^k T(n/2^k) + n\left[\log\left(\frac{n^k}{2^{0+1+2\ldots k-1}}\right)\right]$$

$$= 2^k T(n/2^k) + n\left[\log\left(\frac{n^k}{2^{\frac{k(k-1)}{2}}}\right)\right] \qquad \log(a/b) = \log a - \log b$$

$$= 2^k T(n/2^k) + n\left[\log(n^k) - \log_2\left(2^{\frac{k(k-1)}{2}}\right)\right]$$

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

For B.C

$n/2^k = 1$

$2^k = n$

$k = \log n$

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

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

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

Imp ⇒ Loop Complexity :-

Loop → for, while.

all these statements will run n times

```
for(i=1; i<=n; i++)
{
    S1
    S2
    S3
}
```

O(1) * n = O(n)

AJ(n)

```
i=1
while(i<=n)
{
    S1
    S2
    S3
    i++ → updation.
}
```

How to determine the Time Complexity of a Loop ?:

Time Complexity of any loop depends on two important factors:

1) The number of times the loop is running / iterating / repeated.

AND

2) The Complexity of all the individual statements within it (inside loop body)

eg1:

```
for(j=1; j<=n; j++)
{
    a=a+5
}
```

$a=a+5 \longrightarrow O(1)$

Explanation:-

$j=1 \longrightarrow O(1)$

$j=2 \longrightarrow O(1)$

$\vdots$

$j=n \longrightarrow O(1)$

$O(1)+O(1) \cdots +O(1) = O(n)$ ✓

n times

a = 0

eg2 :- for (i=1; i <= n/2 ; i++)

{ a = a+10 ⟶ O(1)

}

Loop runs n/2 times

} ⟶ O(n/2)

= O(n)

eg3: for(i=1; i<=n; i++)

{ a = a+3

break;

}

n times?

loop only runs 1 time

⇒ O(1) constant time

eg 4:

```
a = 0
for (i=1; i<=n; i++)
  {
    a = a+5
  }
```

a = 0

↓

1st → 5

2nd → 5+5

3rd → 5+5+5

After n iterations

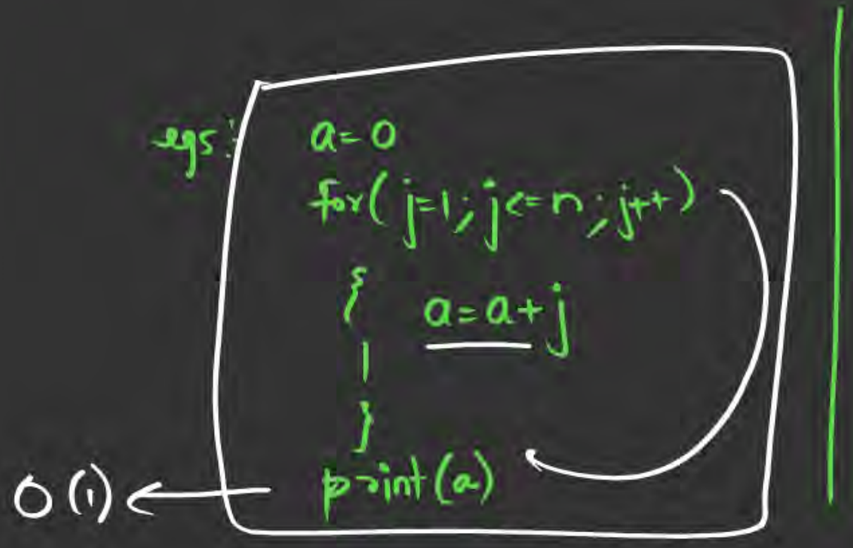
5 * n

(Q.1) What is the TC of code? → O(n)

(Q.2) What is the value of 'a' after code ends?

↓

(5*n)



O(n)

(Q1) TC of given code ?

(Q2) output of give code (in terms of 'n')

a = 0

j=1 ⟶ a = 0+1 = 1 ⟶ 1

j=2 ⟶ a = 1+2 = 3 ⟶ 1+2

j=3 ⟶ a = 3+3 = 6 ⟶ 1+2+3

j=4 ⟶ a = 6+4 = 10 ⟶ 1+2+3+4

⋮

at the end. (after n iterations)

$a = 1+2+3 \cdots + n.$

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

eg:

```
Algo AJ(x,n)
{
    for(i=1; i<=n; i++)
    {
        if(x%i==0)
            break
    }
}
```

(Q) what is the Best and Worst Case Complexity of AJ() & for what type of input?

always runs 1 time $\longrightarrow O(1)$

1st iter: $i=1$

$x\%i==0$

$x\%1$? ✓ $\longrightarrow$ 0

Best case : $x=1$

Worst case : $x=n+1$

$x\%1 \rightarrow$ any int is divisible by 1

eg1

```
Algo AJ(n,n)
{
  for(i=2;i<=n;i++)
    { if(n%i==0)
        { break
        }
    }
}
```

(Q) Time Complexity of AJ() ?

↓

depends on n

W.C

O(n)

If n → prime number

→ O(n) → O(n)

O(n)

If n = 2^2

⇒ O(1) } Best case Ω(1)

eg:-

```
for(i=1; i<=n; i++)
{
    AJ(n)
}
```

Loop Body

$\Rightarrow TC = O(n * O(AJ(n)))$

$i=1 \rightarrow AJ(n)$ — $O(n)$

$i=2 \rightarrow AJ(n)$

$i=3 \rightarrow AJ(n)$ — $O(n)$

$i=n \rightarrow AJ(n)$

1) if $AJ(n) = O(1) \Rightarrow TC = O(n)$

2) if $AJ(n) = O(\sqrt{n}) \Rightarrow TC = O(n\sqrt{n})$

3) if $AJ(n) = O(\log n) \Rightarrow TC = O(n \log n)$

$\rightarrow n * O(n) = O(n^2)$

eg: Given a C program, below:

```
int c=0, i
for(i=1; i<=n; i++);
c=c+i;
```

empty loop Body

$\Rightarrow$

```
for(i=1; i<=n; i++)
{
};
```

terminates when $i = n+1$

Q.1) what is the Time Complexity of given code

$\longrightarrow O(n)$

$\longrightarrow i = n+1$

(Q2) what is the exact value of c after the code ends?

$\longrightarrow c = c+i = 0+(n+1) = \boxed{n+1}$

while loop

```
i=1  //initialization
while(i<=n)        <-- condition
{
    print(i)
    i=i+1 //updation
}
```

(8)

```
i=1, a=0
while (i=2)
{
    a=a+1
}
```

not a condition but its assignment

always True

T(=?

Infinite loop → ∞

a) O(n)

b) O(1)

c) O(logn)

d) None

for + while

$i=1 \longrightarrow O(n/2) \rightarrow O(n)$

$i=2 \longrightarrow O(n)$

$\vdots$

$i=n \longrightarrow O(n)$

$\Big\} \; O(n^2)$

$a=0$

(Q) for ($i=1$; $i<=n$; $i++$) → n times

{

  $j=1$

  while ($j<=n/2$) → $n/2$ times

  {

    $a=a+5$ → runs ($n * n/2$ times) → $\left(5 * n \times \frac{n}{2}\right)$

    $j=j+1$

  }

}

nested loop

1) $T(n) = ?$ ⟶ $O(n^2)$

2) value of a = ? after code ends

H.W

```
a = 0
1) for (i=1; i<=n; i=i+5)
   {
   |   a=a+3
   }
```

Q.1) what is the TC?

(Q2) What is the value of a after code ends?

2 mins Summary

Topic

Topic

Recursive algo TC

Loop Complexity

THANK - YOU

Telegram Link: https://t.me/AdityaSir_PW