Anjali Mishra
CST
04

Page No.
Date / /

Tutorial-2

1. What is the time complexity of below code & how.

```
void fun(int n)
{
    int j=1, i=0;
    while(i<n){
        i = i+j;
        j++; }
}
```

Time complexity - O(sqrt n)

1st time, $i = 1$

2nd time, $i = 3$ $(i = i+2)$

3rd time, $i = 6$ $(i = 1+2+3)$

⋮

nth time $i = \frac{8(x^2+1)}{2} = x^2 < n$

$x = \mathrm{sqrt}(n)$

2. Write recurrence relation for the recursive function that prints Fibonacci series. Solve the recurrence relation to get complexity of the program. What will be the space complexity of this program & why.

Solu2 <u>Recurrence Relation</u>

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

Let $T(n)$ denote the time complexity of $F(n)$.

In $F(n-1)$ and $F(n-2)$, time will be $T(n-1)$ and $T(n-2)$. We have one more addition to sum and results. For $n>1$

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

for $n=0$ and $n=1$, no addition occurs

$\therefore T(0) = T(1) = 0$

Let $T(n-1) \approx T(n-2)$ —— (2)

Adding (2) in (1)

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

Using Backward substitution

$\therefore T(n-1) = 2 \times T(n-2) + 1$

$$T(n) = 2 \times [2 \times T(n-2) + 1] + 1$$
$$= 4T(n-2) + 3$$

We can substitute $T(n-2) = 2 \times T(n-3) + 1$

$\Rightarrow T(n) = 8 \times T(n-3) + 1$

General equation

$$T(n) = 2^k \times T(n-k) + (2^k - 1) \quad (3)$$

For $T(0)$:

$n - k = 0$

$\Rightarrow k = n$

Substituting values in (3)

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

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

Space complexity $\rightarrow O(n)$

Reason

The function calls are p execute sequentially. Sequential execution guarantees that the stack size will not exceed the depth of calls. For first $F(n-1)$ it will create 'n' stack frames, the other $F(n-2)$ will create

N/2. so, the longest ~~de~~ is 'n'.

3. Write programs which have complexity - n(logn), n^3, log(log n)

→ For time complexity : n^3
We can use three nested loops - $O(n^3)$

```
for(int i=0; i<n; i++)
{
    for(int j=0; j<n; j++)
    {
        for(int k=0; k<n; k++)
        {
            some O(1) expression
        }
    }
}
```

⇒ For time complexity - log(log n)
We can use the following function

```
for(int i=2; i<n; i=pow(i,k))
{
    //some O(1) expression
}
```

where 'k' is constant

→ For time complexity $n\log n$

We can use the following function

```
int fun(int n)
{
    for(i=1; i<=n; i++)
    {
        for(j=1; j<=n; j+=i)
        {
            some O(1) expression
        }
    }
}
```

4. Solve the following recurrence relation

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

<u>Soln</u> $T(n) = 2T(n/2) + cn^2$

using master's method

$T(n) = aT(n/b) + f(n)$

$a \geq 1,\ b > 1,\ c = \log_b a$ [comparing n^c & $f(n)$]

We get, $c = \log_2 2 = 1$

$f(n) > n^c$

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

$\Rightarrow \theta(n^2)$

5. What is the time complexity of the following function

```
int func(int n)
{
    for(int i=1; i<=n; i++)
    {
        for(int j=1; j<n; j+=i)
        {
            some O(1) task
        }
    }
}
```

solu for $i=1 \longrightarrow j=1,2,3,4 \text{-----} n$ (sum for 'n' times)

for $i=2 \longrightarrow j=1,3,5 \text{------}$ (sum for $n/2$ times)

for $i=3 \longrightarrow j=1,4,7 \text{-----}$ (sum for $n/3$ times)

$$T(n) = n + n/2 + n/3 + n/4 + \text{----}$$

$$= n(1 + 1/2 + 1/3 + 1/4 + \text{-----})$$

$$= n\int_1^n \frac{1}{x} \Rightarrow n\int_1^n \frac{dx}{x} \Rightarrow \log x\Big]_1^n$$

$$= n\log n$$

∴ The Time complexity of following function is $n\log n$.

6. What should be the time complexity of following function

```
for ( int i=2; i<n; i=pow(i,k))
{
    // some O(1) expressions or statem
}
```

where 'k' is constant

Sol: for first iteration, $i=2$

2nd iteration, $i = 2^k$

3rd iteration, $i = (2^k)^k$

⋮

nth iteration, $i = 2^{k^i}$

Loop ends at $2^{k^i} = n$

Apply log, $\log n = \log 2^{k^i} \Rightarrow k^i = \log n$

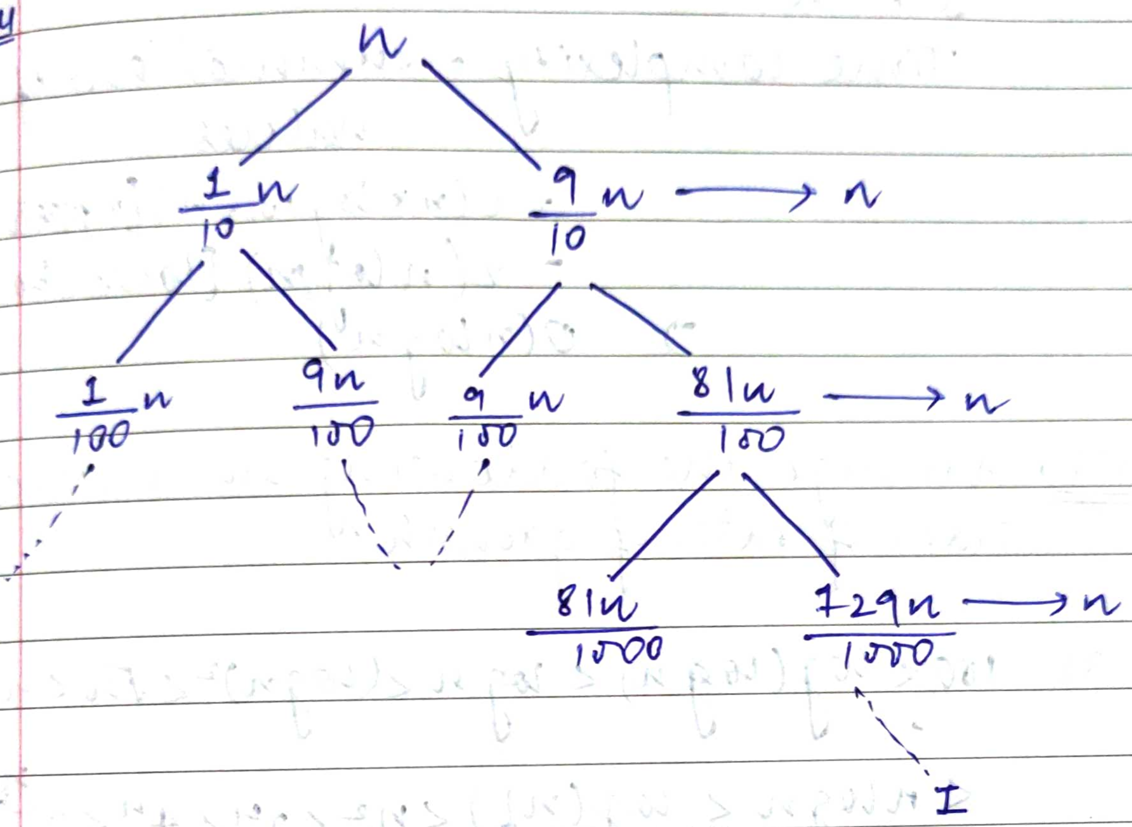
Again apply log $\log(k^i) = \log n$

$\Rightarrow i = \log_e(\log n)$

7. Write a recurrence relation when Quick Sort repeatedly divides the array in to two parts of 99% & 1%. Drive the time complexity in this case. Show the recursion tree while driving time complexity & find the difference in height of both the extreme paths. What do you understand

by this analysis?

Soln



If we split in this manner:

Recurrence relation →

$$T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + O(n)$$

When first drawn is of size 9n/10 and second one is n/10. sorting the above using recurrsion tree approach calculating values

At 1st level, value = n

At 2nd level, value $= \frac{9n}{10} + \frac{n}{10} = n$

Value remains same at all levels i.e. n

Time complexity = summation of values

$= O(n \times \log_{10/9} n)$ [upper bound]

$= \Omega(n \log_{10} n)$ [lower bound]

$\Rightarrow O(n \log n)$

Ques 8. Arrange the following in increasing order of rate of growth.

a) $100 < \log(\log n) < \log n < (\log n)^2 < \sqrt{n} < n$
$< n\log n < \log(n!) < n^2 < 2^n < 4^n < 2^{2^n}$

b) $1 < \log(\log n) < \sqrt{\log(n)} < \log n < \log 2n$
$< 2(\log n) < n < n(\log n) < 2n < 4n < \log(n!)$
$< n^2 < n! < 2^{2^n}$

c) $96 < \log_8 n < \log 2n < 5n < n\log_6 n < n(\log_2 n)$
$< \log(n!) < 8n^2 < 7n^3 < n! < 8^{2n}$