Tutorial → 2

Q1 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++;
    }
}
```

Sol $i = 0 + 1 + 2 + 3 + 4 \cdots\cdots + K,$

$$i = \frac{K(K+1)}{2} = \frac{K^2+K}{2}$$

now $i < n$ so $\frac{K^2+K}{2} < n$

$K^2 < n$

$K < \sqrt{n}$

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

Q2 Find time complexity of Fibonaci series by recurrence relation ?

```
int fun (int n)
{
    if (n <= 1)
        return (n);

    return fun (n-1) + fun (n-2);
}
```

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

$T(n-1) \simeq T(n-2)$

$$T(n) = 2T(n-2) + c \quad -①$$

put $n = n-2$ in eq ①

$$T(n-2) = 2T(n-4) + c \quad - ②$$

put eq ② in eq ①

$$T(n) = 2[2T(n-4) + c] + c$$
$$= 4T(n-4) + 3c \quad - ③$$

put $n = n-4$ in eq ①

$$T(n-4) = 2T(n-6) + c \quad - ④$$

put eq ④ ino eq ③

$$T(n) = 4[2T(n-6) + c] + 3c$$
$$= 8T(n-6) + 7c.$$

$$\Rightarrow 2^k T(n-2k) + (2^k - 1)c$$

$$n - 2k = 1$$
$$2k = 1 + n$$
$$k = \frac{1+n}{2} = \frac{n}{2}$$

$$T(n) = 2^{n/2} T(n - \cancel{2}\tfrac{n}{\cancel{2}}) + (2^{n/2} - 1)c$$
$$= 2^{n/2} + (2^{n/2} - 1)c$$
$$= 2^{n/2} + 2^{n/2}c - c$$
$$= 2^{n/2}(1 + c) - c$$

// Constant Ignored

$$= 2^{n/2}$$
$$T(n) = O(2^{n/2})$$

Space Complexity → The space is proportional to the maximum depth of the recursion

Hence the space complexity of Fibonacci Series in recursive is $O(N)$.

Q3 Write program which have time complexity
(i) $n(\log n)$ (ii) n^3 (iii) $\log(\log n)$

Sol (i)

```
for (int i=1 ; i<=n; i++)
{
  for (int j=1; j<=n ; j=j*2)
  {
     sum = sum + j;
  }
}
```

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

(ii)

```
for (int i = 1 ; i <= n ; i++)
{
  for (int j = 1; j <= n ; j++)
  {
     for (int k = 1; k <= n ; k++)
     {
        sum = sum + k;
     }
  }
}
```

$T(n) = O(n^3)$

(iii) for(i = 1 ; i <= n ; i * = 2)

{

for (j = i ; j >= 1 ; j = j/2)

{

O(1)

}

}

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

Q4 $T(n) = T(n/4) + T(n/2) + cn^2$

Using Master's Theorem

we can assume $T(n/2) >= T(n/4)$

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

$a = 2, b = 2 \Rightarrow c = \log_b a$

$c = \log_2 2 = 1$

$f(n) > n^c$

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

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

Q5 int fun (int n)

{

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

{ for (int j =1 ; j < n ; j += i)

{ // some O(1) task

}

} }

for $i=1$, inner loop is executed n times
for $i=2$, " " is " $n/2$ times
for $i=3$ " " is " $n/3$ times

it is forming a series

$\Rightarrow \quad n + \frac{n}{2} + \frac{n}{3} + \cdots \frac{n}{n}$

$n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}\right)$

$= n \times \sum_{K=1}^{n} \frac{1}{K}$

$= n \times \log n$

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

Q6 for (int i=2; i<=n; i = pow(i, k))
{
// some O(1) expression.
}

wit iteration :-

for 1st iteration $\rightarrow 2$

for 2 " $\rightarrow 2^K$

for 3 " $\rightarrow (2^K)^K$

for n iteration $\rightarrow 2^{K\log_K(\log n)}$

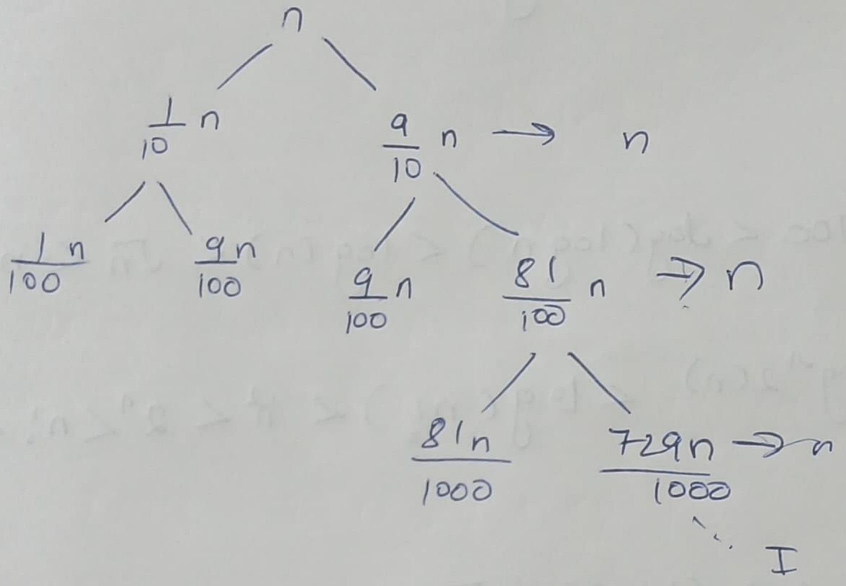
$\Rightarrow 2^{\log_2 n} = n$ $\quad$ $\boxed{a^{\log_a b} = b}$

Each iteration takes constant times

$\therefore$ Total iteration $= \log_k(\log n)$

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

Q7



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 branch is of size $9n/10$ & second one in $n/10$. Solving the above using recursion tree approach calculating values

~~At~~ At 1^{st} level, value $= n$

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

value remains same at all level i.e.

Time complexity = Summation of values

$= O(n \times \log n)$ upper bound

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

$T(n) = O(n \log n)$ //.

Q8

(a) $100 < \log(\log n) < \log(n) < \sqrt{n} < n < n\log$

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

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

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

(c) $96 < \log_8(n) < n\log_6 n < \log_2 n < n\log_2 n < \log(n!)$

$< 5n < 8n^2 < 7n^3 < n! < (8)^{2n}$.