Samya Jain
40 (I)

Tutorial - 2

(1)

Que-1. what is the time Complexity of below code and how?

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

$i = 0, 1, 3, 6, 10, 15$

Let us say k terms.

So, general form would be $\frac{k(k+1)}{2}$

k^{th} term $= n \Rightarrow \frac{k(k+1)}{2} = n$

$k^2 = 2n \Rightarrow k = \sqrt{n}$

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

Que-2. Write Recurrence relation for the recursive function that prints fibonnacci series. solve the recurrence relation to get the time complexity of this program and why?

Sol^n-2. Recurrence Relation →

Recursive function

```
int fib( int n)
{ if (n<=1)          → O(1) = c
    return n;
  return fib(n-1) + fib(n-2)   → T(n-1) + T(n-2)
}
```

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

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

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

By backward Substitution.

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

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

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

$T(n) = 4T(n-2) + 3c$

Now, $T(n-2) = 2T(n-2-1) + c$
$= 2T(n-3) + c$

$\therefore T(n) = 4T(n-2) + 3c$
$= 4(2T(n-3) + c) + 3c$
$T(n) = 8T(n-3) + 7c$
$\vdots$

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

assume $n-k=0 \Rightarrow n=k$
$2^n T(0) + (2^n - 1)c$
$= 2^n + (2^n - 1)c$
$= 2^n(1+c) - c$
$= 2^n$
$T(n) = O(2^n)$ Ans

Space complexity - for fibonacci recursion implementation, the space required is directly proportional to the maximum depth of recursion tree, since maximum depth is directly proportional to number of element so $O(n)$.

Que-3. Write programs which have $T(n) \rightarrow n(\log n)$.

Ans-3. 1
```
for (i=1; i<=n; i++)
{ for (j=1; j<=n; j*2)
  { sum = sum + i;
  }
}
```

2 n^3

```
for (i=0; i<n; i++)
{ for (j=0; j<n; j++)
{ for (k=0; k<n; k++)
{ sum = sum + k;
}}}
```

Now, we know that $n\left(1+\frac{1}{2}+\frac{1}{3}+ \cdots +\frac{1}{n}\right) \le n\left(1+\frac{1}{2}+\frac{1}{2}+\frac{1}{4}+\frac{1}{4}+\frac{1}{4}+ \cdots\right)$

$$n\left(1+\frac{1}{2}+\frac{1}{3}+ \cdots +\frac{1}{n}\right) \le n\left(1+0.5+0.5+ \cdots\right)$$

$O(n \log n)$ Ans.

Ques-6. What should be the time complexity of -

```
for (int i=2 ; i<=n; i= pow (i,k))
{ // some O(1) expression or statements
} where k is constant.
```

Ans-6

for first iteration $i=2$

for second iteration $i = 2^k$

for third iteration $i = (2^k)^k = 2^{k^2}$

⋮

n^{th} iteration, loop ends when $2^{k^i} = n$

Take log on both sides

$\log n = \log 2^{k^i}$

$\log n = k^i \Rightarrow \boxed{i = \log(\log n)}$

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

Ques-7. Write a recurrence relation when quick sort repeatedly divides the array in two parts of 99% and 1%. Derive the time complexity in this case. Show the recurrence tree while deriving time complexity and find the difference in heights of both the extreme parts. What do you understand by the analysis?

Sol^n 7 99 to 1 is quick sort, when pivot is where from front or end always, so;

$$T(n) = T(99n/100) + T(n/100) + O(n)$$

$$
\begin{tikzcd}[column sep=small]
 & & & T(n) \arrow[dll] \arrow[drr] & & & \\
 & T(99n/100) \arrow[dl] \arrow[dr] & & & & T(n/100) \arrow[dl] \arrow[dr] & \\
T\left(\frac{(99)^2}{100} \times n\right) & & T(99n/(100)^2) & & T(99n/(100)^2) & & T(n/(100)^2)
\end{tikzcd}
$$

$$n = \left(\frac{99}{100}\right)^k$$

$$\log n = k \log \frac{99}{100}$$

$$k = \log_n \frac{100}{99}$$

$$\therefore \text{Time complexity} = n * \log\left(\frac{100}{99} n\right)$$ Ans

Que-8. Arrange the following in increasing order of rate of growth.

a) $n, n!, \log n, \log\log n, \text{root}(n), \log(n!), n\log n, \log^n(2n), 2^n, 2^{2^n}, 4^n, n^2, 100$

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

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

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