Tutorial-2

Q)

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

'j' can be defined according to the relax on $i_j = i_{j-1} + j$. The value of j inc. by one for each i know on is the sum of first 'i' position increases of the total no. of iterations taken by program, the while loop term notes if

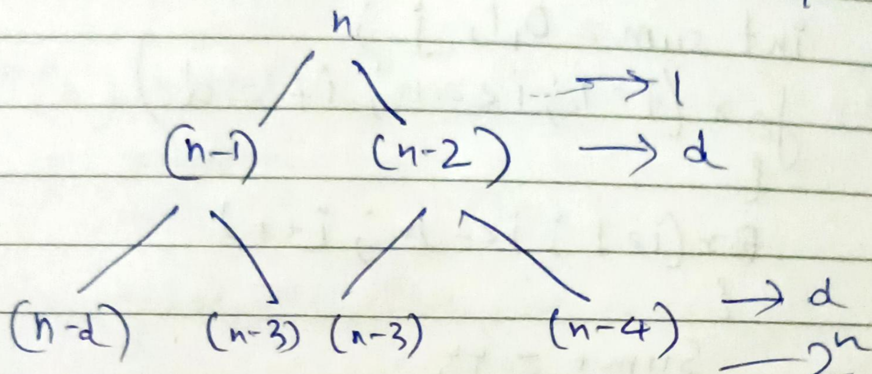
$$1+2+3+---+k = \left[\frac{k(k+1)}{2}\right] > n$$

$$\text{so } k = O(\sqrt{n})$$

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

Q6 Recurrence relation for recursive function that prints fibonnaci series is

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



$$T = 1 + 2 + 4 + \dots + \cancel{\alpha}\, n$$

$$a = 1 \qquad \cancel{r} = \frac{2}{1} = 2$$

$$\frac{a(r^{n} - 1)}{r - 1} = 1\left(\frac{2^{n+1} - 1}{2 - 1}\right)$$

$$= 2^{n+1} - 1$$

$$O(2^{n+1}) = O(2^n \cdot 2)$$

$$T.C. = O(2^n)$$

S.C. is $O(n)$ because there are 'n' no. of functions calls during recursions without using recursion it will be $O(1)$

Q) Program having Complexity
n (log n)

```
int sum = 0, i, j;
for (i = 1; i <= n; i * = 2)
{
for (j = 1; j <= n; j++)
{
Sum + = j;
}
}
```

ii) n^3

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

iii) log (log n)

```
int i = 2, count = 0, j;
i = i * 2;
while (j < n)
{
```

```
count++;
j += i;
}
```

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

~~$T(n/2)$~~

$T(n/2) >= T(n/4)$

equal on can be.

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

Using master's method,

$a = 2, b = 2$

$c = \log_2 2 = 1$

$n^c = n$

$f(n) = n^2$

$f(n) > n^2$

$T.C. = O(f(n)) = O(n^2)$

Q)

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

$TC = O(n^2)$

6 ~~for (int i=~~

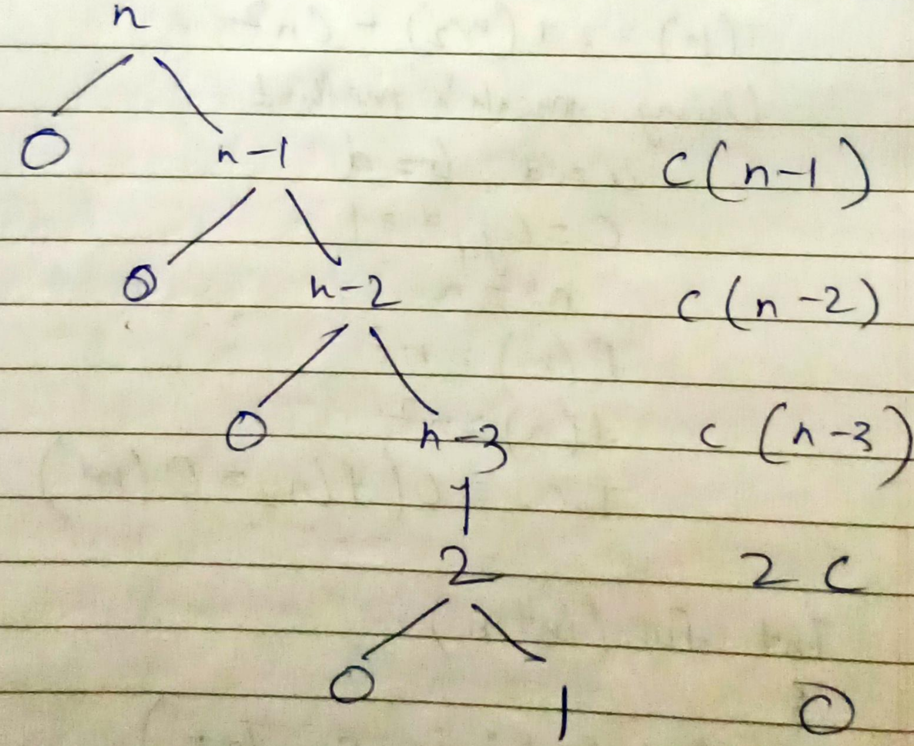
where k is constant

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

7 Recurrence relation will be

$T(n) = T(9n/10) + T(n/10) + O(n)$

Subproblem size   Total partition



$Cn + C(n-1) + C(n-2) + 2C = C\left(\frac{(n+1)n}{2} - 1\right)$

$O(n^2)$