Tutorial 2

1.

i	j
0	1
1	2
3	3
6	4
10	5
⋮	⋮

No. of times loop is running be k

$S_k = 1 + 3 + 6 + 10 + \dots + T_k$

$S_{k-1} = 1 + 3 + 6 + \dots + T_{k-1}$

subtracting both

$S_k - S_{k-1} = 1 + 2 + 3 + 4 + \dots + (k-1)$

$$T_k = \frac{(k-1)\,k}{2}$$

Given that k^{th} term is n.

$T_k = n$

$\frac{k(k-1)}{2} = n \Rightarrow \frac{k^2}{2} - \frac{k}{2} = n$ Ignoring lower order terms & constants

$\Rightarrow k^2 = n$

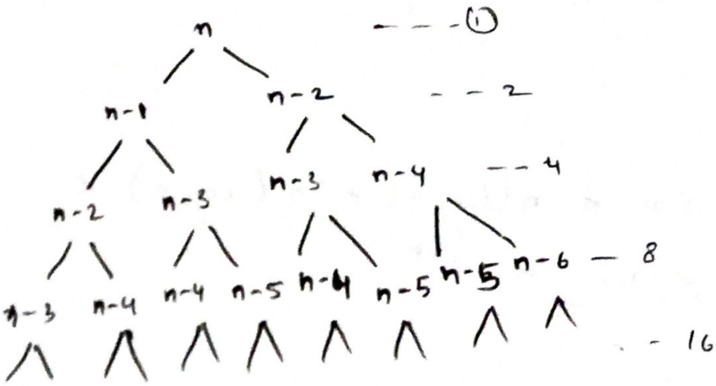
$\Rightarrow k = \sqrt{n}$

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

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

For recursive fibonacci solution

Recursion Tree :



No. of times function is running will be sum of the series :

$S = 1 + 2 + 4 + \dots + 2^n$

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

Time Complexity :

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

After removing constants

From recursion tree, height of tree = n

$\Rightarrow \boxed{\text{Space Complexity} = O(n)}$

3\. Code having time complexity :

$O(n \log n)$:

```
for ( int i=1; i<=n; i++)
{         for (int j=1; j<n; j=j*2)
                printf (" Hello ");

}
```

$O(n^3)$:

```
for (int i=0; i<n; i++)
{        for (int j=0; j<n; j++)
         {         for (int k=0; k<n; k++)
                         printf ("Hello");

         }

}
```

$O(\log(\log n))$:

```
for (int i=2; i<=n; i=pow(i,3)))
{        printf ("Hello");

}
```

* Here n can be any positive integer .

4. $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + cn^2$

Ignoring lower order terms :

$$T(n) = T\left(\frac{n}{2}\right) + cn^2$$

Using Master Theorem's

$a = 1$, $b = 2$, $f(n) = n^2$

$C = \log_b a = \log_2 1 = 0$

$\boxed{0 < n^2}$ True

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

5.

i	j
1	n
2	n/2
3	n/3
4	n/4
⋮	⋮

Time complexity will be sum of series

$$S = \frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \cdots$$

$$= \sum_{i=1}^{n} \left(\frac{n}{i}\right)$$

$$\text{Complexity} = n \times \sum_{i=1}^{n} \left(\frac{n}{i}\right)$$

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

Sequence :

$$2,\ 2^k,\ (2^k)^k,\ ((2^k)^k)^k \ \ldots\ldots$$

Generalising : $2^{k^0},\ 2^{k^1},\ 2^{k^2},\ \ldots\ldots,\ 2^{k^{\lambda-1}}$

<u>Assumption</u> : Let no. of terms be λ.

Given : Last term is n.

$$\Rightarrow \quad 2^{k^{\lambda-1}} = n$$

$$k^{\lambda-1} \log 2 = \log n$$

$$k^{\lambda-1} = \log n \qquad \text{[Ignoring constant } (\log 2)]$$

$$(\lambda-1)\log k = \log(\log n)$$

$$\boxed{\lambda = \log(\log n)} \qquad (\text{Ignoring constant terms})$$

Time Complexity : $\boxed{T(n) = O(\log(\log n))}$

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