Tutorial-2

(1)

i	j
0	1
1	2
3	3
6	4
10	5

No. of times loop is running is K.

$S_K = 1+3+6+10+ \dots T_K$

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

Substracting 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.

$\Rightarrow K^2 = n$

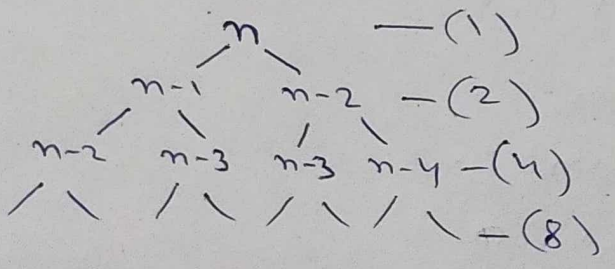
$\Rightarrow K = \sqrt{n}$

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

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

For recursive fibonacci solution.

Recursive 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:

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

From Recursive Tree
Height of Tree = n

] After removing constants

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

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

Ignoring lower order terms:

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

Using Master Theorem,

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

$c = \log_b a - \log_2 1 = 0$

$0 < n^2$ 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:

$S = n + \frac{n}{2} + \frac{n}{3} + \dots$

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

Complexity $= n \times \sum_{i=1}^{n} \left(\frac{n}{i}\right) \Rightarrow \boxed{T(n) = n \log n}$

⑥ Sequencing :

$2, 2^k, (2^k)^k, ((2^k)^k)^k, - - -$

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

Assumption : Let no. of terms be λ.

Given : Last term is n

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

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

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

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

$$\boxed{\lambda = \log(\log n)} \quad \text{[Ignoring constant terms]}$$

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

⑧ a) $100 < \log(\log n) < \log n < (\log n)^2 < \sqrt{n} < n < n(\log n)$
$< \log(n!) < n^2 < 2^n < 4^n < 2^{2^n}$.

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

b) $6 < \log_8 n < \log 2n < 5n < n(\log_6 n < n(\log_2 n)$
$< \log(n!) < 8n^2 < 7n^3 < n! < 8^{2n}$.