Tutorial : 2

1)

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

Ans 1st time $i=1$

2nd time $i=3$

3rd time $i=6$

⋮

nth time $i = x^2 \leq n$

$\Rightarrow x = \sqrt{n}$

$O(\sqrt{n})$ Ans

2) $fib(n) = fib(n-1) + fib(n-2)$ Let $T(0)=1$

fib(n) : if $n<=1$
   return 1.
   return $fib(n-1) + fib(n-2)$

Time complexity:

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

$$= 2T(n-2) + C \quad \{T(n-1) \cong T(n-2)\}$$

$$T(n-2) = 2*(2T(n-2-2)+C)+C$$

$$= 4T(n-4) + 3C$$

$$T(n-4) = 2*(4T(n-4+1)+3C)+C$$

$$= 8T(n-3) + 7C$$

Scanned by TapScanner

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

$n - k = 0 \Rightarrow n = k.$

$T(n) = 2^n * T(0) + (2^n - 1)c$

$= 2^n * 1 + 2^n c - c$

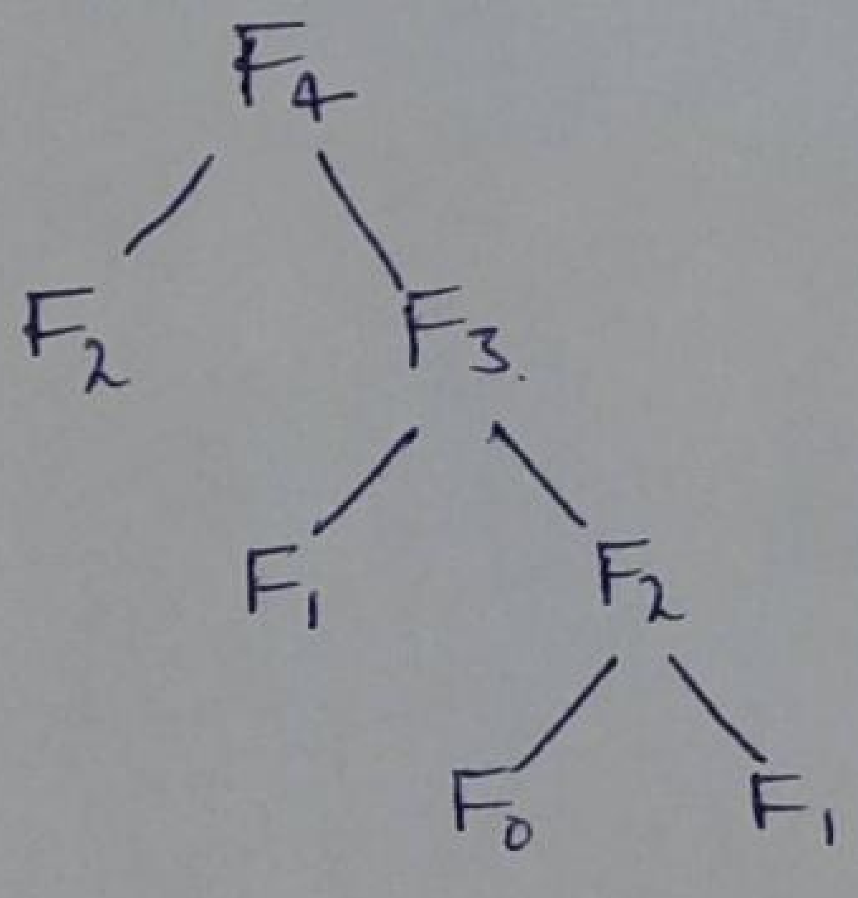
$= 2^n (1 + c) - c$

$\equiv 2^n$ // constant can be ignored.

$O(2^n)$ Ans

Space Complexity: The space is proportional to the maximum depth of the recursion tree.

(Hence, the space complexity of Fibonacci recursive is $O(N)$)



3)

$n \log n$

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

n^3.

```
int arr[n1][n2][n3];
for(int i=0; i<n1; i++)
{
  for(int j=0; j<n2; j++)
  {
    for(int k=0; k<n3; k++)
    {
      printf(arr[i][j][k]);
} } }
```

log(log n)

```
for(int i=2; i<n; i = pow(i,k))
{
    // some O(1) expressions as statements
}
```

4) $T(n) = T(n/4) + T(n/2) + Cn^2$

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

Using Master's Method, $T(n) = aT\left(\frac{n}{b}\right) + Cn^2$

$a = 2, b = 2$ $\quad f(n) = Cn^2$

$n^c = n^{\log_2 2} = n$

$n^c < f(n)$

$\Rightarrow$ $\Theta(n^2)$ Ans

5)
```
int fun(int n){
        fun(int i=1; i<=n; i++)
    {
     for(int j=1 ; j<n; j+=i)
     {
      // some O(1) task }
     }
    }
```

Ans when $i=1$ inner loop j will run n times

" $i=2$ inner loop j will run $n/2$ times

" $i=3$ inner loop j will run $n/3$ times

⋮

when $i=n$ " " " " " n/n times.

Scanned by TapScanner

So, $n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}$

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

$= n \log n$

~~or~~ $O(n \log n)$ Ans

6) for(int i=2; i<n; i = pow(i,k))

{

// some O(1) expressions as statements

}

where k is a constant.

⇒ as 1st ⇒ $i = 2, 2^k, 2^{k^2}, 2^{k^3}, \ldots\ldots 2^{k^i}$

⇒ $2^{k^i} = n$

taking log on both sides.

$k^i = \log n$

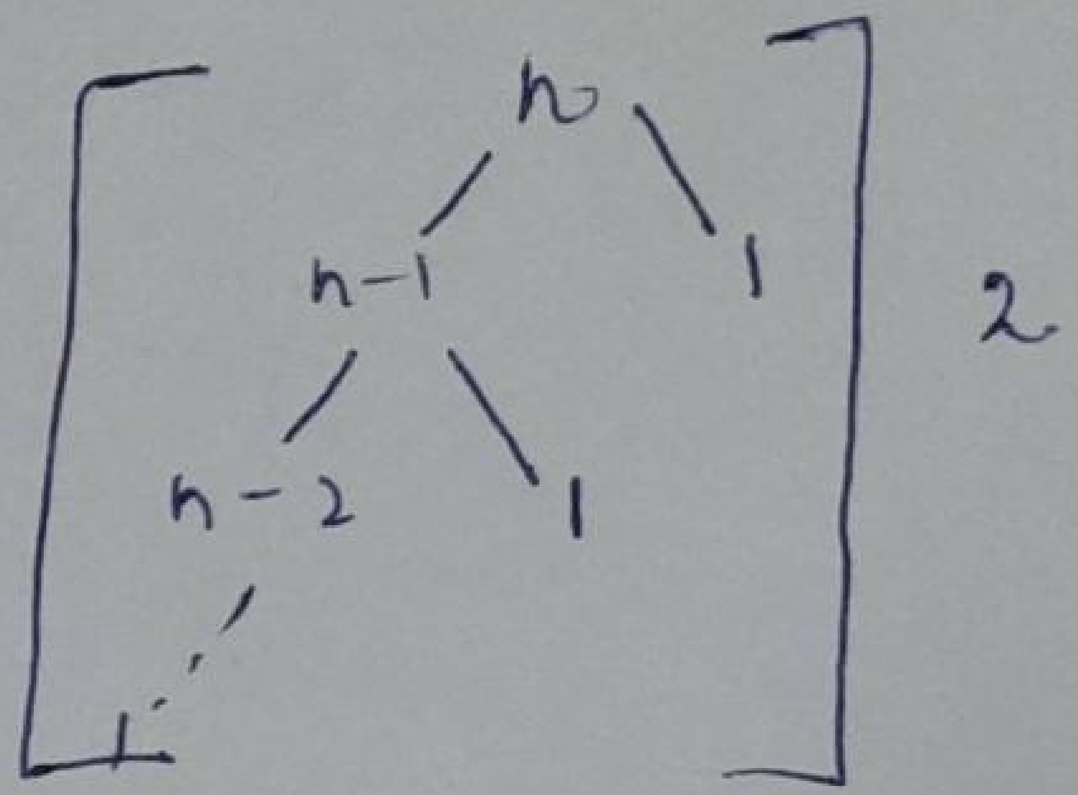
$i \log_k k = \log(\log n)$

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

So, time complexity $= \log(\log n) * O(1) = \boxed{\log(\log n)}$ Ans.

7) array is divided into 99% & 1%.

$\because T(n) = T(n-1) + O(1)$



$T(n) = (T(n-1) + T(n-2) + \ldots + T(1) + O(1)) \times n$

$= n \times n.$ n level.

$\because T(n) = O(n^2)$

Lowest height = 2.

Highest height = n

$\therefore \boxed{\text{diff} = n-2}$ $n > 1$.

The given algorithm provides linear results.

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

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

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

1 7.