

① Void fun (int n) {
 int j = 1, i = 0; $\rightarrow O(1)$
 While (i < n)
 {
 i = i + j;
 j++;
 }
}

$i = 0 \rightarrow i = 0 + 1 = 1, j = 2$
 $i = 1 \rightarrow i = 1 + 2 = 3, j = 3$
 $i = 3 \rightarrow i = 3 + 3 = 6, j = 4$
 $i = 6 \rightarrow i = 6 + 4 = 10, j = 5$
 \vdots
 $i = k \rightarrow i = \underbrace{k + x}_{\text{Sum}}, j = n$

From the above pattern we can clearly see that it is just a sum of first n - numbers.

OR

j	1	2	3	4	...	n
i	1	3	6	10	...	k

$$\frac{k(k+1)}{2} > n$$

dominating term

$$\frac{k^2 + k}{2} > n$$

$$k^2 = n$$

$$k = \sqrt{n}$$

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

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

Soln. 2

```
int fib (int n)  $\longrightarrow$   $T(n)$ 
{
    if ( $n \leq 1$ )  $\longrightarrow O(1)$ 
        return n;
    return fib( $n-1$ ) + fib( $n-2$ );  $\longrightarrow T(n-1) + T(n-2)$ 
}
```

$$T(n) = \begin{cases} 1 & , n \leq 1 \\ T(n-1) + T(n-2) & , \text{otherwise.} \end{cases}$$

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + C \\ &= 2T(n-1) + C \end{aligned}$$

[From the approximation $T(n-1) \sim T(n-2)$]

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

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

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

\vdots

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

We know that $T(1) = 1$, so

$$T(n-k) = T(1) = 1$$

$$n-k = 1$$

$$\boxed{k = n-1}$$

We get

$$T(n) = 2^{n-1} T(1) + (2^{n-1} - 1)C$$

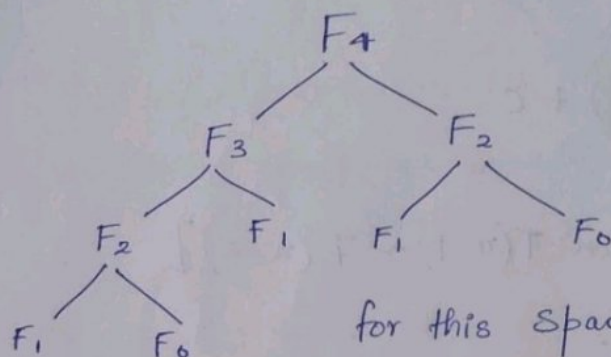
$$T.C = O(2^n)$$

Now, for space complexity we know that

Space reqd. \propto Max. depth of recursive tree.

because that is the max. no. of elements that can be present in the implicit function call stack.

now, let's take an example F_4 , so



for this space comp. = $O(4)$

for n -elements, $S.C = O(n)$

Soln-3 (i) for $T.C = n \log n$.

$A()$

```
{ int i, j;
```

```
  for (i=1; i<=n; i++)  $\rightarrow O(n)$ 
```

```
  { for (j=1; j<=n; j=j*2)  $\rightarrow O(\log n)$ 
```

```
    { printf("Hey");
```

```
    }
```

```
  }
```

```
}
```

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

now, $T(n/4) \leq T(n/2)$

So we can write this equation as

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

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

Comparing with Master equation we get

$$a=2, b=2, k=2 \text{ \& } p=0$$

now $\boxed{a < b^k}$ so, $\neq a < b^k \text{ \& } p=0$,
 $2 < 2^2$

$$T.C = \Theta(n^k \log^p n)$$

$$= \Theta(n^2 (\log n)^0)$$

$$\boxed{T.C = \Theta(n^2)}$$

Soln. 5 int fun (int n) {

for (int i=1; i<=n; i++)

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

{ // Some O(1) task

}}}

j incrementation depends on i . \therefore we unroll all loops.

i = 1	i = 2	i = 3	-----	i = n
j = 1 to n	j = 1 to n	j = 1 to n		j = 1 to n
"n-times"	"n/2 times"	"n/3 times"		"1" times

So,

$$T.C = n + \frac{n}{2} + \frac{n}{3} + \dots + \frac{n}{n}$$

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

$\log n$

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

Soln. 6

```
for (int i = 2; i <= n; i = ik)
{
    // Some O(1) stmt
}
```

$i = 2$	$i = 2^k$	$i = (2^k)^k = 2^{k^2}$
"2 to n times"	"2 ^k to n times"	"2 ^{k²} to n times"
	$2^{k \log_k \log n}$	

at that time, $2^{k \log_k (\log n)} = 2^{\log n} = n$

$$\text{Total } T.C = O(\log \log n)$$

OR

for ~~k~~

$i = 2$	$i = 2^k$	$i = 2^{k^2}$	\dots	$i = 2^{k^l}$
		$2^{k^l} = n$		

$$k^l = \log_2 n$$

$$l = \log_{k^2} \log_2 n$$

$$T.C = O(\log_k \log_2 n)$$

(ii) for, $T.C = n^3$

A()

```
{ int i, j, k;
```

```
  for (i=0; i<=n; i++)  $\rightarrow O(n)$ 
```

```
  { for (j=0; j<=n; j++)  $\rightarrow O(n)$ 
```

```
    { for (k=0; k<=n; k++)  $\rightarrow O(n)$ 
```

```
      { printf("Hey");
```

```
      }
```

```
    }
```

```
  }
```

```
}
```

(iii) $T.C = \log(\log n)$