```
Factorial(n) {                  T(n) = T(n-1) + 3 for n≥0 (3 Units of cost of the simple operations)
    If n==0 then return 1       T(0) = 1
    return n x Factorial(n-1)
}
```

Back Substitution

T(n) = T(n-1) + 3 … 1
T(n-1) = T(n-2) + 3 … 2
T(n-2) = T(n-3) + 3 … 3

Substitute 2 in 1 T(n) = T(n-2) + 6 … 4
Substitute 3 in 4 T(n) = T(n-3) + 9 … 5

For generic k T(n) = T(n-k) + 3k … 6

Base/Anchor condition: n-k = 0 ⇒ n=k
T(n) = T(0) + 3n = 1 + 3n … *O*(n)

Space Complexity: In case of n=5, The stack is F(5), F(4),F(3),F(2),F(1) which means without declaring any variable. The depth is the stack is 5 which is n in this case. So, Space complexity is *O*(n). This is in terms of recursive calls ONLY and does not take the accounting information into account.

$$\mathrm{T(n)} \begin{cases} n * T(n-1) & ; n > 0 \\ 1 & ; n = 0 \end{cases}$$

```
A(n) {                          T(n) = c + 2T(n/2) where c is a constant time
    If( … )
        return A(n/2)+A(n/2)
}
```

```
void test(int n) {              T(n)
    if( n>0 ) {                 Exit clause is n=0
        cout << n;              1 unit of time
        test(n-1);              T(n-1)
    }
}
```

T(n) = T(n+1) + 1

$$\mathrm{T(n)} \begin{cases} T(n-1) + 1 & ; n > 0 \\ 1 & ; n = 0 \end{cases}$$
