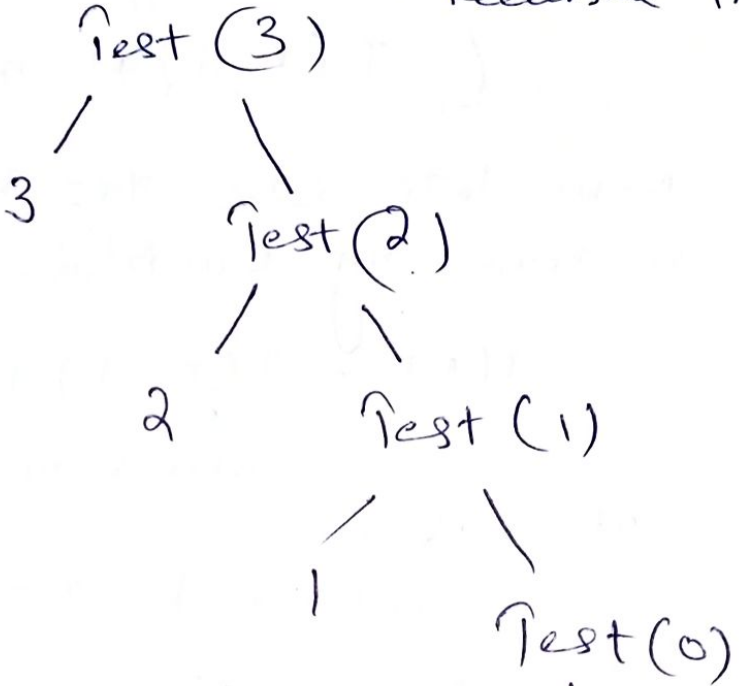
Recursion

```
void Test (int n)
{
  if (n>0)
  {printf("%d", n)
   Test (n-1)
  }
}
```

e.g Pass (3) → Tracing Tree
→ Recursive Tree



Time taken by Test = 3 + 1 (x)

if n = 5 then 5 + 1 => (n + 1)

& printf = n times

f(n) = n + 1 (Time Complexity)

Recurrence Relation

⇒ Time Taken by function $T(n)$ ↓ Tells time Taken by recursive fn

ignore condition

for print = 1

for Test(n-1) = (n-1)

$\rightarrow T(n)$
void Test()
1 — if n>0
1 — print
n-1 Test(n-1)

Total time = $T(n-1) + 1$

If we consider condition = $T(n-1)+2$ ↓ Constant

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

→ Now lets solve this recurrence relation by substitution method

$T(n) = T(n-1)+1$ ← (1)
↓
what is this.

As we know

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

~~T(n) = T(n-1+1)+1+1~~

$\because T(n-1) = T(n-2)+1$

$T(n) = [T(n-2)+1]+1$

$= T(n-2)+2$

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

⋮

The pattern is (2)

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

$$T(n) = T(n-k) + k \quad \text{--- (iv)}$$

When it becomes $n = 0$ at (iv)

So $n - k = 0$.

then $n = k$.

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

$$T(n) = T(0) + n$$

$T(0) = 1$

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

$$\Downarrow$$

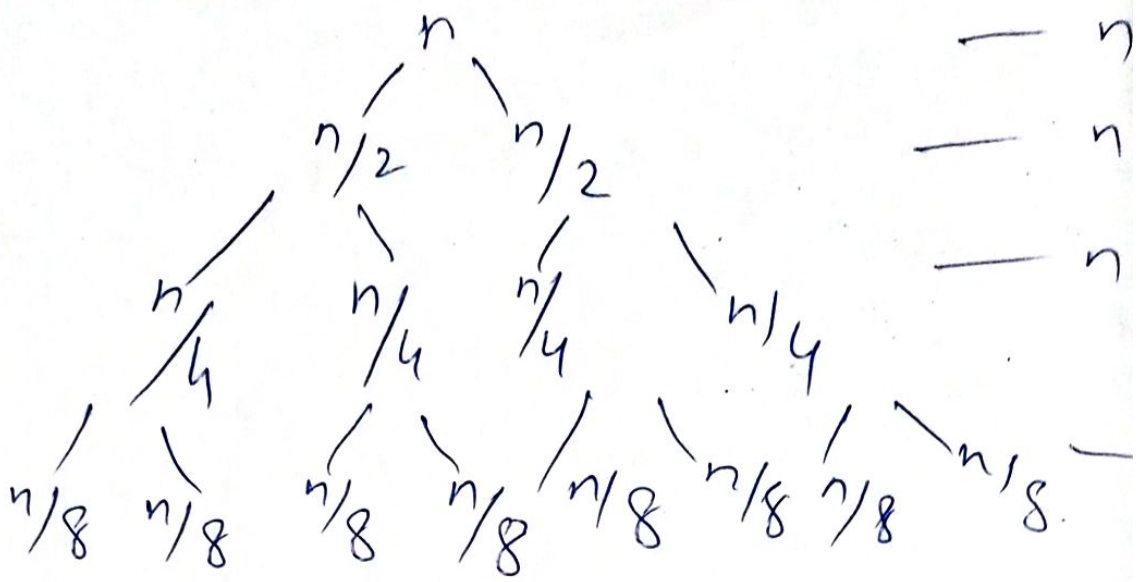
$O(n)$ or $\Theta(n)$ or

```
void Test (int n)  ——  T(n)
{
  if (n>1)
  {
    for (i=0 ; i<n ; i++)  ] - n
    {
      Statement
    }
    Test (n/2);  —  T(n/2)
    Test (n/2);  —  T(n/2)
  }
```

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

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + n & n > 1 \end{cases}$$

Dividing whole is usually a solution is seen



$\Rightarrow 2^k$.

$n/2k \quad n/2k \quad n/2k \quad \ldots \quad n/2^k$

So at all levels = n

how many ~~times~~ times k this

It will apprech 1. $= nk \quad -(i)$

So $\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log(n), \quad -(ii)$

$= n\log(n)$

$O: n\log(n)$

$$T(n) \begin{cases} 1 & n=1 \\ 2T(n/2)+n & n>1 \end{cases}$$

$$T(n) = 2T(n/2)+n \quad -(1)$$

(ii) $T(n/2) = 2T(n/2^2)+\frac{n}{2}$

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

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

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

$$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right)+\frac{n}{2^2}$$

$$= 2^2 T\left(2\frac{n}{2^3}+\frac{n}{2^2}\right)+2n$$

$$= 2^3 T\left(\frac{n}{2^3}\right)+n+2n$$

$$= 2^3 T(n/2^3)+3n$$

If k times

$$2^k T(n/2^k)+kn.$$

Assume $T\left(\frac{n}{2^k}\right) = T(1)$

$$\frac{n}{2^k} = 1$$

$$k = \log n$$

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

As $2^k = n$

$k = \log n$

$= 2^k T(1) + kn$

$= n \cdot 1 + n \log n$

higher order is $n \log n$

So $O(n) = n \log(n)$