Recursive Analysis

Recursion : A function calling itself is called "Recursion"

↳ Recursion is preferable when, In a given problem every subproblem is same as given problem.

For Recursive analysis space complexity is more compare to normal programs. for every recursive call again it will create stack.

In order to analyse Time complexity of Recursive function is we have various methods they are :

1. Back substitution Method
2. Recursive Tree Method
3. Master theorem

Eg:

```
fun() {
    statement 1
    :
    statement n
    fun()
    :
}
```

Recursive fun call

Steps:

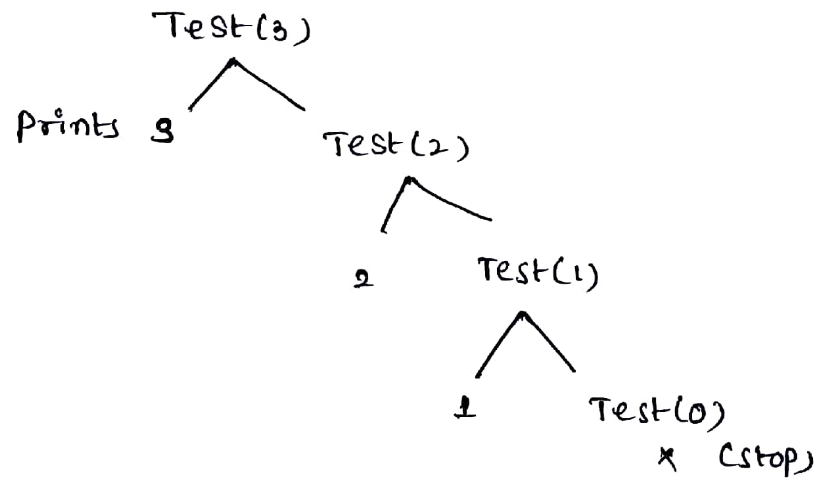
1. In order to Analyse any Recursive program we need to write Recurrence relation.
2. By solving that recurrence relation only we will get Time complexity.

Eg1:

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

Treemethod:

Let $n=3$



The amount of work done here → printing value + calling function

As I passed '3' → 3 times printing If

I passed '5' → 5 times prints

for each call it is printing 1 unit of time so.

n.of calls $3+1=4$

n.of prints -3

In general for any 'n' — n.of calls '$n+1$'

n.of prints 'n'

Time function $f(n) = O(n)$

$\boxed{\therefore \text{ Time Complexity } = O(n)}$

Eg1:
```
void Test(n) {            —— T(n)
    if (n>0)              ——> O(1)
    {
        P.f("%d", n);     ——> 1
        Test(n-1);        ——> T(n-1)
    }
}
```

Step1 : Write down recurrence relation, for if and printf statement it will take constant amount of time and again it will call Test(n-1) so the relation is

$$T(n) = T(n-1) + 1 \quad [\because \text{ as if and printf takes } O(1)]$$

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

Step2: Solve Recurrence relation by Back substitution method.

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

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

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

$$\vdots$$

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

consider 1 $\quad T(n) = 1 + T(n-1)$

substitute $T(n-1)$ value from 2

$$\Rightarrow \quad T(n) = 1 + T(n-1)$$

$$T(n) = 1 + \overbrace{1 + T(n-2)} = 2 + T(n-2)$$

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

$$\vdots$$

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

when it will stop whenever $n-k=0 \Rightarrow n=k$

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

Substitute $n = k$

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

$T(n) = T(0) + n \Rightarrow 1 + n$

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

Eg 2:

```
void Test (n)                              <-----------
{    if (n>0) {  --> 1                                |
       for (i=0; i<n; i++) -> n+1                     |
       { pf ("%d", n); } n                            |  recursive call
       Test (n-1);  -------------------------------------
         |-> T(n-1)
}
```

Step 1:

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

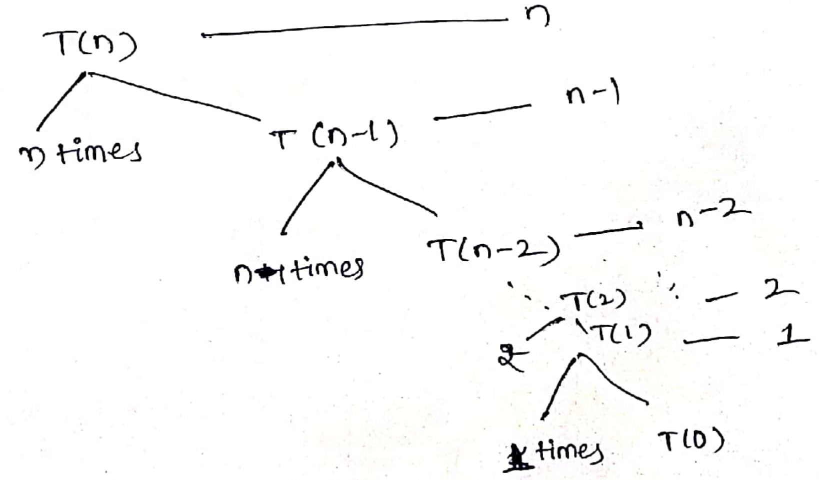
$= T(n-1) + 2n+2$

writing it as Asymptotically $O(n)$

So reccurrence relation now,

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

Tree method:



Total time is:

$$T(n) = \underbrace{n + (n-1) + (n-2) + \dots 2+1}_{\text{sum of 1st natural numbers}}$$

$$T(n) = \frac{n(n+1)}{2} \simeq O(n^2)$$

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

<u>Backsubstitution Method:</u>

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

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

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

$$\vdots$$

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

from 1

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

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

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

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

$$\vdots$$

ktimes

$$= T(n-(k+1)) + n-k + \dots T(n-3) + (n-2) + (n-1) + n$$

$$= T(n-k) + (n-(k-1)) + n-(k-2) + \dots$$

$$\dots + (n-1) + n$$

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

$$T(n) = T(0) + (n-n+1) + (n-n+2) + \dots (n-1) + n$$

$$= 1 + \underbrace{1 + 2 + 3 + \dots (n-1) + n}$$

$$= n(n+1)/2 \simeq O(n^2) \quad //$$

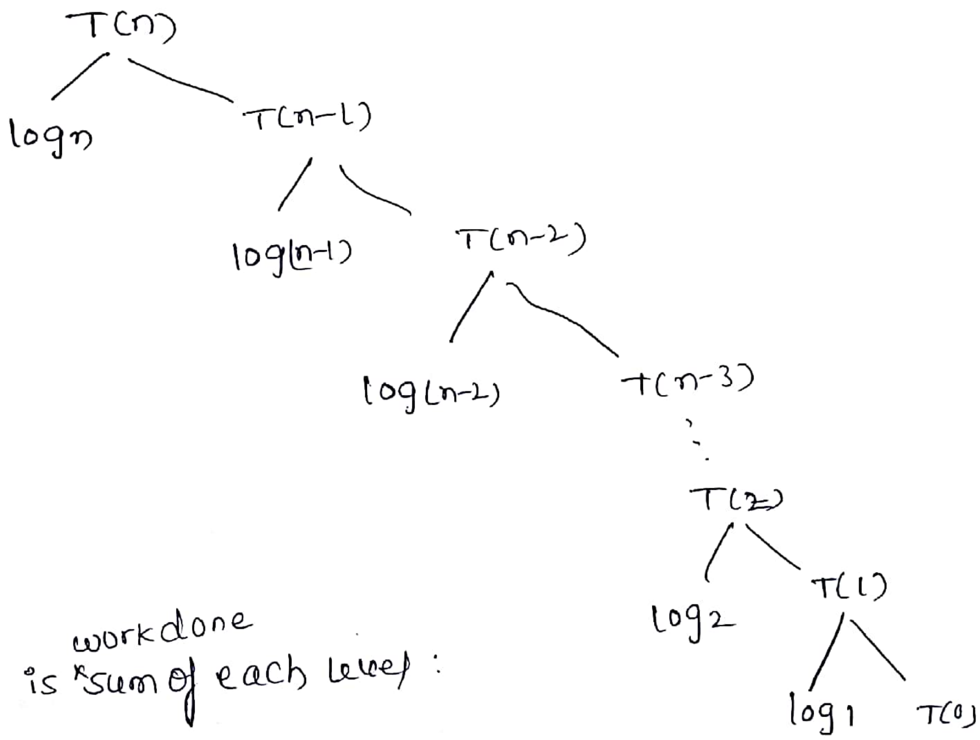
Eg3:

```
void Test(n)
{    if (n>0)                       ——— O(1)
     {    for (i=1; i<n; i*2)   }
               pf("%d", i);     } — O(log n)
     Test(n-1)        ——>  T(n-1)
     }
```

write Recurrence relation : $T(n) = T(n-1) + \underbrace{\log n + O(1)}$

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

Treemethod:



∴ Time taken is ^{x}sum of each level : (workdone)

$T(n) = \log n + \log(n-1) + \log(n-2) + \cdots \log 1$

$\log[\underbrace{n * (n-1) * (n-2) * (n-3) * \cdots}\ 1)$

$= \log[n!]$ $\quad [\because n! \simeq n]$

$= \log n^n$ $\quad$ There is no bound so upperbound $n! \simeq n^n$

$\boxed{T(n) \Rightarrow O(n \log n)}$

Back substitution Method :

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

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

$$= \underline{T(n-2)} + \log(n-1) + \log n$$

$$= [T(n-3) + \log(n-2)] + \log(n-1) + \log n$$

$$= T(n-3) + \log(n-2) + \log(n-1) + \log n$$

After 'k' times $\vdots$

$$T(n) = T(n-k) + \log 1 + \log 2 + \log 3 + \cdots \log(n-1) + \log n \rightarrow (1)$$

we know

$n - k = 0$

$n = k \rightarrow$ substitute in (1)

$$T(n) = T(0) + \underbrace{\log 1 + \log 2 + \log 3 + \cdots + \log n}$$

$$= 1 + \log n!$$

$$= 1 + \log n^n$$

$$= 1 + n\log n$$

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

Scanned by CamScanner

Conclusions:

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

$T(n) = T(n-1) + n \longrightarrow O(n^2)$

$T(n) = T(n-1) + \underline{\log n} \rightarrow O(n \log n)$

↓

fun is multiplied by 'n' times

Instead by $T(n-1)$ we can take $T(n-2)$

Then also same thing.

$T(n) = T(n-\underline{2}) + 1 \rightarrow n/2 \simeq O(n)$

$= T(n-\underline{100}) + 1 \rightarrow O(n)$

↓

constant

∴ In general we can write from above observations

If any recurrence relation in the form of

$$\boxed{T(n) = T(n-\underset{\substack{\downarrow \\ \text{constant}}}{k}) + \underset{\substack{\downarrow \\ \text{Asymptotic fun}}}{f(n)}}$$

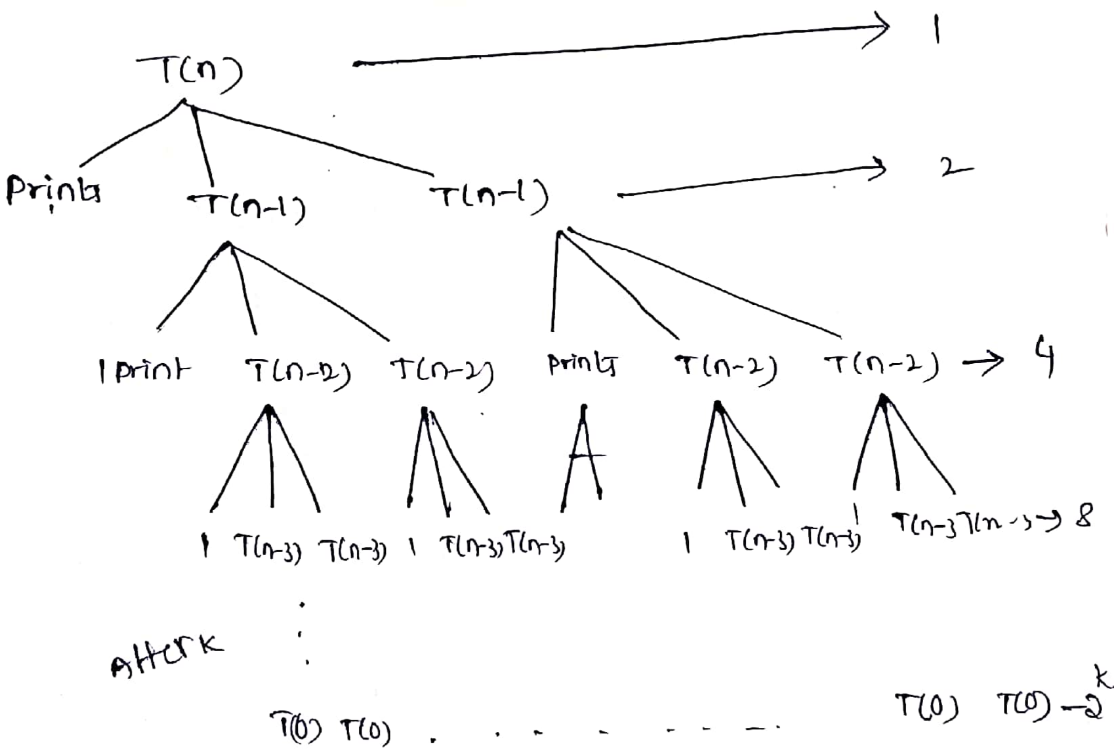
Then Time complexity is n times of $f(n)$

Exe:

```
void Test (n) {              -> T(n)
    if (n>0)                 -> 1
    { pf("%d",n);            -> 1
      Test(n-1);             -> T(n-1)
      Test(n-1);             -> T(n-1)
    }
}
```

$$T(n) = \begin{cases} 2T(n-1) + 1 & n>0 \\ 1 & n=0 \end{cases}$$

Using Recursion Tree method :



Total work done sum of works in each level.

$1 + 2 + 2^2 + 2^3 + \dots \; 2^k$ [∵ sum of G.P series]

$a = 1$ $\quad a + ar + ar^2 + \dots \; ar^k$

$r = 2$

$$\text{sum} = \frac{a(r^{k+1} - 1)}{r - 1} = 2^{k+1} - 1 \simeq \boxed{O(2^n)}$$

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

Back substitution Method:

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

~~T(n-1)~~ $= 2\left[2T(n-2)+1\right] + 1$

$$= 2^2 T(n-2) + 2 + 1 \rightarrow (2)$$

$$= 2^2 \left[2T(n-3)+1\right] + 2 + 1$$

$$= 2^3 T(n-3) + 2^2 + 2 + 1 \longrightarrow (3)$$

$\vdots$

After K times

$$T(n) = 2^k T(n-k) + 2^{k-1} + 2^{k-2} + \cdots 2^3 + 2^2 + 2 + 1$$

Assume $n-k=0$

$n=k$

substitute 'k' value as 'n'

$$T(n) = 2^k T(0) + \underbrace{2^{k-1} + 2^{k-2} + \cdots 2^3 + 2^2 + 2 + 1}$$

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

$$\Rightarrow 2^n + 2^n - 1 \Rightarrow 2^{n+1} - 1$$

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

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

//

Ex5: Test(n) { $\longrightarrow$ $T(n)$

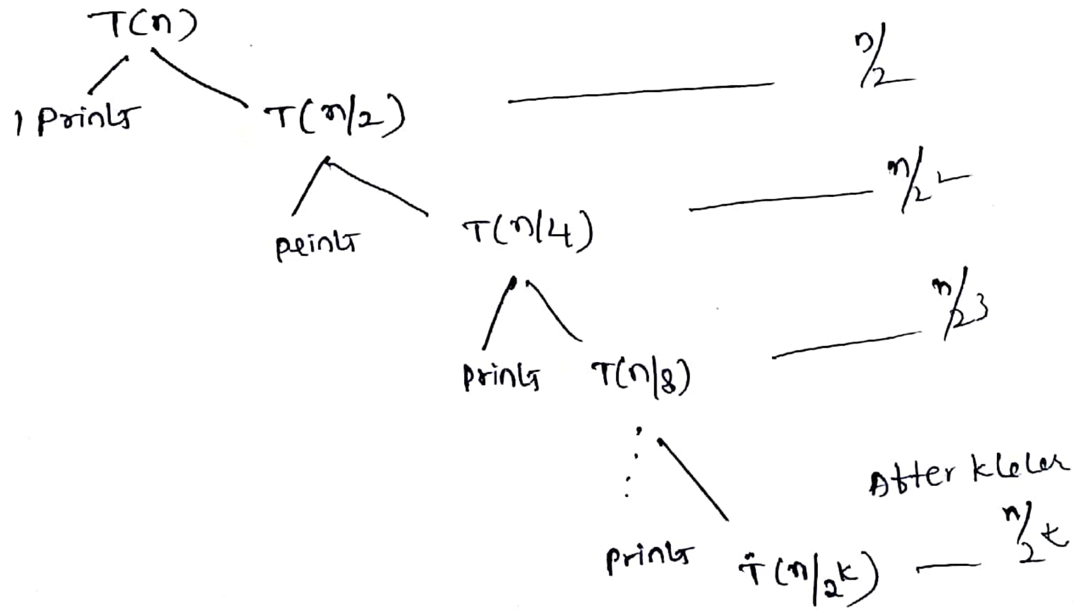
```
if (n>1)
{ pf ("%d", n);      -> 1
  Test (n/2);        -> T(n/2)
}
}
```

$$T(n) = T(n/2) + 1$$

Recurrence relation:

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

Recursion Tree method:



we know it will stop at

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

$$n = 2^k$$

$$k = \log n$$

It will stop after k level so Total time is:

$$1 * k = 1 * \log n = O(\log n)$$

Back substitution Method:

$$T(n) = T(n/2) + 1 \rightarrow (1)$$

$$T(n/2) = [T(n/2^2) + 1] + 1$$

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

$$= [T(n/2^3) + 1] + 2$$

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

⋮

After k :

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

$$[\because n/2^k = 1]$$
$$2^k = n$$
$$k = \log n$$

$$= T(1) + k$$

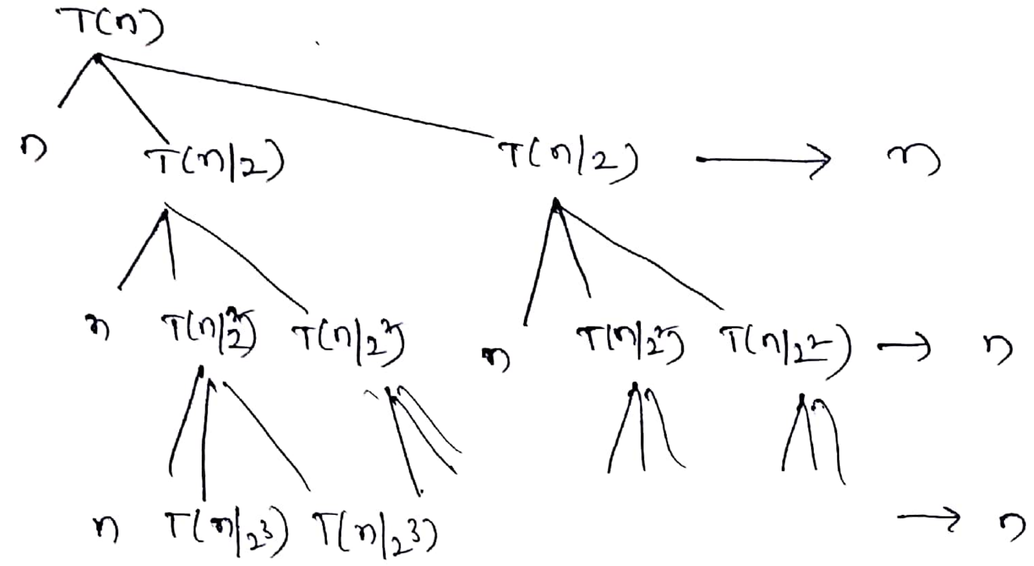
$$T(n) = 1 + \log n$$

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

Eg6:

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

Reccursive Tree method:



After
k level

$T(n/2^k)$

Total work done After k levels is $\Rightarrow k * n$

we know it will stop when $n/2^k = 1$

$$n = 2^k$$

$$k = \log n$$

Substitute

$$T(n) = k * n$$

$$= \log n * n$$

$$\boxed{\therefore\ T(n) = O(n\log n)}$$

Recurrence relation:

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

Scanned by CamScanner

Back substitution Method:

$$T(n) = 2T\left(\frac{n}{2}\right) + n \rightarrow (1)$$

$$\Rightarrow 2\left[2T(n/2^2) + n/2\right] + n$$

$$\Rightarrow 2^2 T(n/2^2) + n + n$$

$$\Rightarrow 2^2 T(n/2^2) + 2n \longrightarrow (2)$$

$$\Rightarrow 2^2\left[2T(n/2^3) + n/2^2\right] + 2n$$

$$\Rightarrow 2^3 T(n/2^3) + 3n \longrightarrow (3)$$

$\vdots$

After k^{th} recursive calls

$$\Rightarrow 2^k \underbrace{T(n/2^k)} + kn$$

$$n/2^k = 1 \Rightarrow k = \log n$$

$$\Rightarrow 2^{\log n} T(1) + \log n * n$$

$[2^k = n = 2^{\log n}]$

$$\Rightarrow n + n\log n$$

$$\boxed{\therefore T(n) = O(n\log n)}$$

Eg7:

```
Test(n) {
if (n>2)              → 1
  { statement;        → 1
    Test(√n)          → T(√n)
  }
}
```

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

Using Back substitution method:

$T(n) = T(n^{1/2}) + 1 \rightarrow (1)$

$= [T(n^{1/2^2}) + 1] + 1$

$= T(n^{1/2^2}) + 2 \rightarrow (2)$

$= T(n^{1/2^3}) + 3 \rightarrow (3)$

After k:

$T(n^{1/2^k}) + k$

Assume $n = 2^m$

$T(2^m) = T(2^{m/2^k}) + k$

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

$2^k = m$

$k = \log_2 m$

$= \log_2 m$

$\Rightarrow \log_2 \log n$

$n = 2^m$

$m = \log n$

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

Scanned by CamScanner