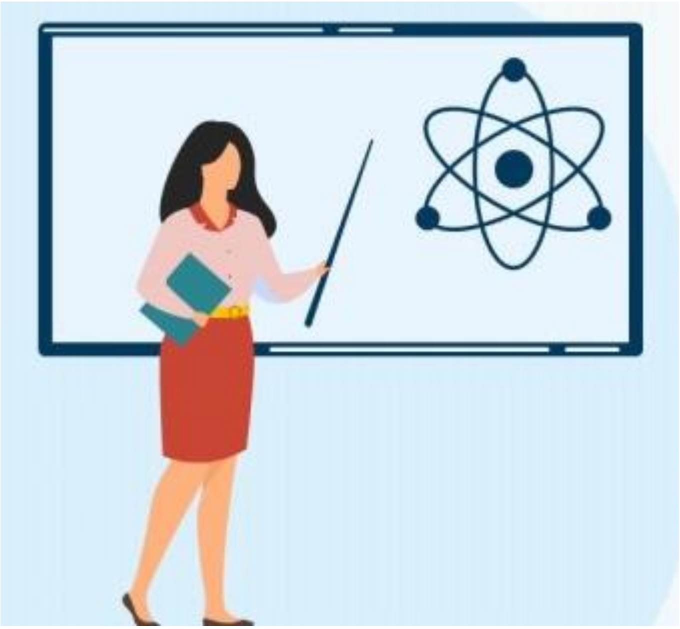
Recurrence Relation



Suman Pandey

Agenda

- **Recursion Tree Method**
- **Substitution Method**
- **Masters Theorem**
- Recurrence Relation ($T(n) = T(n-1) + 1$)
- Recurrence Relation ($T(n) = T(n-1) + n$)
- Recurrence Relation ($T(n) = 2\,T(n-1) + 1$)
- **Masters Theorem Decreasing Function**
- Recurrence Relation Dividing Function
- Recurrence Relation ($T(n) = 2*T(n/2) + 1$)
- Recurrence Relation ($T(n) = 2*T(n/2) + n$)
- **Masters Theorem in Algorithms for Dividing Functions**

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

In recurrence relation, we usually call the function as T(n). As it is a time function

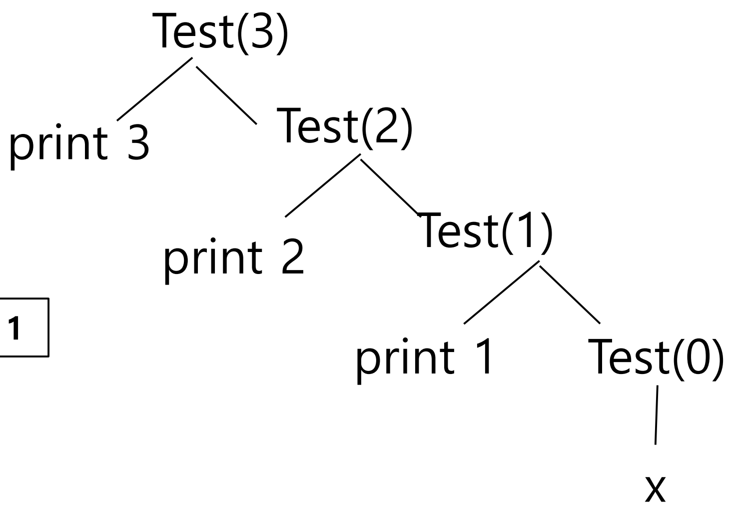
Finding Time complexity using **Recursive Tree**

What is the time complexity of this function ?

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

Test (3)
```

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



F(n) =(n + 1) -> there are n +1 function calls, and each function call takes one unit of time to print

Time Complexity - > Θ(n)

Substitution Method (T(n) =T(n-1) +1)

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

Test (3)
```

T(n) -> It is total amount of time taken by this function

if (n >0) → 1 (one unit)

printf("%d",n); → 1 (one unit)

Test(n - 1); → T(n -1)

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

When n is 0, the time it takes is 0, but we don't write it 0, we make is a constant time 1, you can take any constant

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

$T(n) = T(n-1) + 1$
$= [\ T(n-2) + 1\ ] + 1 = T(n-2) + 2$
$= [\ T(n-3) + 1\ ] + 2 = T(n-3) + 3$
$= [\ T(n-4) + 1\ ] + 3 = T(n-4) + 4$
.....
.....
.....
$= T(n-k) + k$

This substitution can go upto n = 0
Hence we assume that $n - k = 0$
which means $n = k$

$T(n) = T(n-n) + n$
$T(n) = T(0) + n$
$T(n) = 1 + n$

Time Complexity - Θ(n)

Time complexity following of this Recurrence relation using recursion tree or substitution method is coming to be same as O(n)

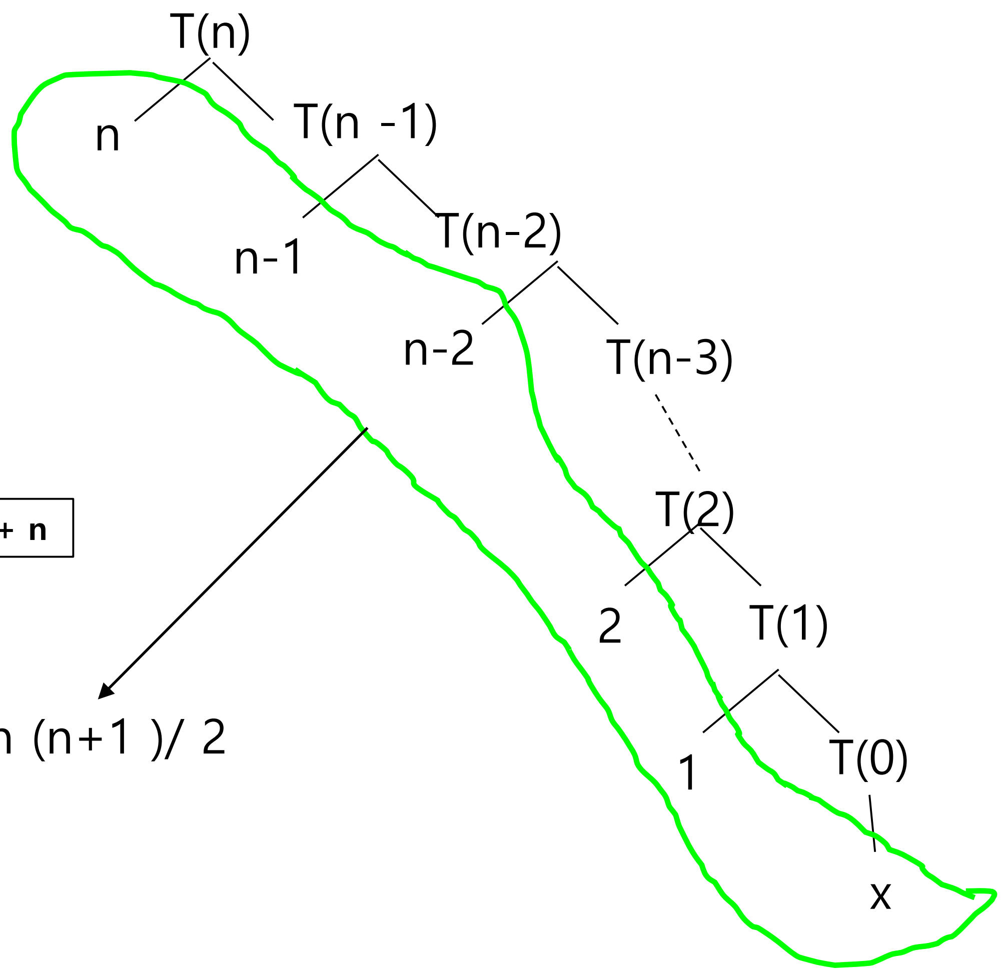
Recurrence Relation ($T(n) = T(n-1) + n$)

Finding Time complexity using **Recursive Tree**

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

```
Void Test( int n )                 ---->  This function takes T( n )
{
   if ( n >0 )                     ---->  1 unit time
   {
      for ( i=0 ; i< n ;i++)       ---->  n +1 (unit)
      {
          printf("%d",n );         ---->  n (unit)
      }
      Test(n - 1);                 ---->  T( n -1 )
   }
}
```

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



1+ 2 + 3 ……. + n-1 + n = n (n+1)/ 2

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

Time Complexity = Θ(n²)

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

```
Void Test( int n )
{
   if ( n >0 )
   {
      for ( i=0 ; i< n ;i++)
      {
          printf("%d",n );
      }
      Test(n - 1);
   }
}
```

This function takes T(n)

1 unit time

n +1 (unit)

n (unit)

T(n -1)

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

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

Note: avoid adding two n's keep it as a sequence, as you need to com e up with a formula at the end

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= [\,T(n-2) + n - 1\,] + n = T(n-2) + (n-1) + n \\ &= [\,T(n-3) + n - 2\,] + (n-1) + n = T(n-3) + (n-2) + (n-1) + n \\ &= [\,T(n-4) + n - 3\,] + (n-2) + (n-1) + n \\ &= T(n-4) + (n-3) + (n-2) + (n-1) + n \\ &\;\;\ldots\ldots \\ &\;\;\ldots\ldots \\ &\;\;\ldots\ldots \\ &= T(n-k) + (n-(k-1)) + (n-(k-2)) + \ldots\ldots + (n-1) + n \end{aligned}$$

This substitution can go upto $n = 0$

Hence we assume that $n - k = 0$

which means $n = k$

$$\begin{aligned} T(n) &= T(n-n) + (n-n+1) + (n-n+2) + \ldots\ldots\ldots + (n-1) + n \\ T(n) &= T(0) + 1 + 2 + 3 + \ldots\ldots\ldots + (n-1) + n \\ T(n) &= T(0) + n(n+1)/2 \\ T(n) &= 1 + n(n+1)/2 \end{aligned}$$

Time Complexity - $\Theta(n^2)$

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

Finding Time complexity using **Recursive Tree**

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

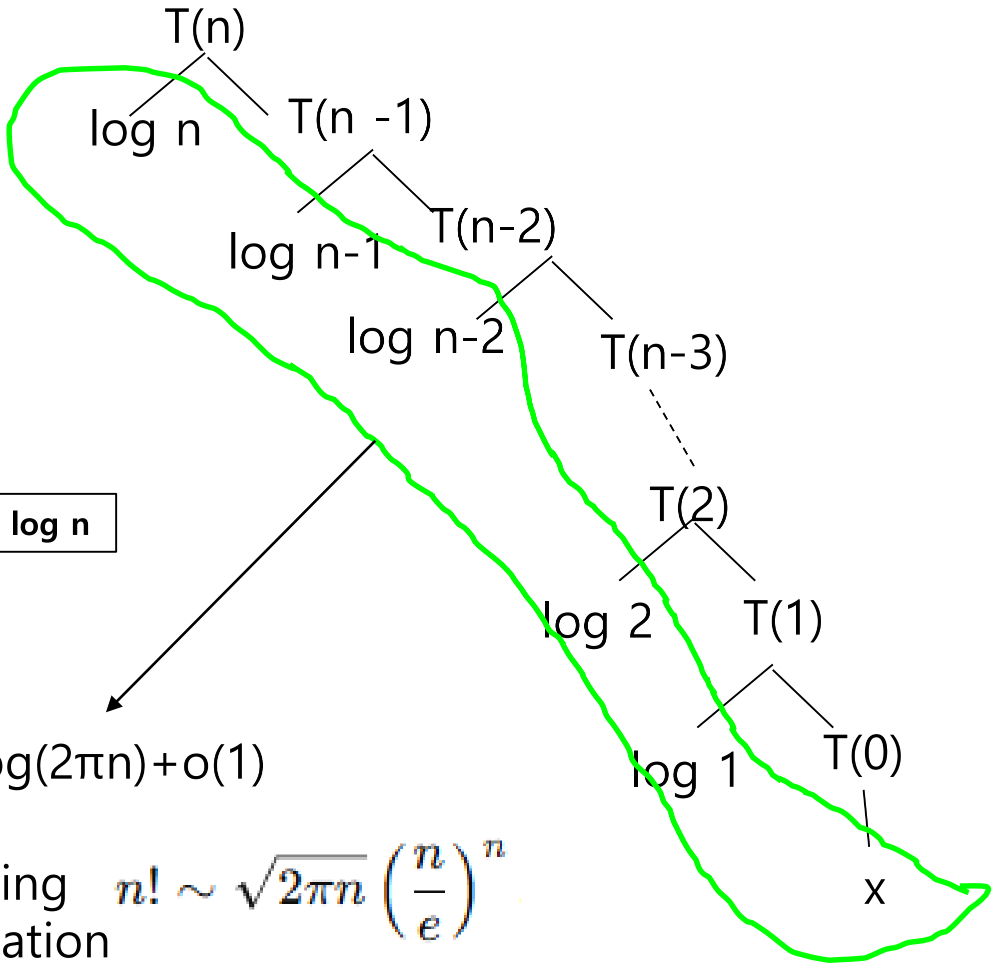
```
Void Test( int n )
{
   if ( n >0 )
   {
      for ( i=1 ; i< n ;i=i*2)
      {
          printf("%d",n );
      }
      Test(n - 1);
   }
}
```

Void **Test**(int n) → This function takes T(n)

for (i=1 ; i< n ;i=i*2) { printf("%d",n); } → log n (unit)

Test(n - 1); → T(n -1)

T(n) = T (n -1) + log n - > T (n -1) + log n



log1+ log2 + log3 ……. + log(n-1) + log(n)

= log(1x2x…..x(n-1)(n)) = log n! = n log n − n + 1/2log(2πn)+o(1)

Stirling equation $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

Time Complexity = Θ(n log n)

Substitution Method (T(n) =T(n-1) +log n)

```
Void Test( int n )
{
    if ( n >0 )
    {
        for ( i=0 ; i< n ;i=i*2)
        {
            printf("%d",n );
        }
        Test(n - 1);
    }
}
```

This function takes T(n)

log n (unit)

T(n -1)

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

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

$$\begin{aligned}
T(n) &= T(n-1) + \log n \\
&= [\, T(n-2) + \log(n-1) \,] + \log(n) = T(n-2) + \log(n-1) + \log n \\
&= [\, T(n-3) + \log(n-2) \,] + \log(n-1) + \log n \\
&\qquad = T(n-3) + \log(n-2) + \log(n-1) + \log n \\
&= [\, T(n-4) + \log(n-3) \,] + \log(n-2) + \log(n-1) + \log n \\
&\qquad = T(n-4) + \log(n-3) + \log(n-2) + \log(n-1) + \log n \\
&\quad \ldots\ldots \\
&\quad \ldots\ldots \\
&\quad \ldots\ldots \\
&= T(n-k) + \log(n-(k-1)) + \log(n-(k-2)) + \ldots\ldots + \log(n-1) + \log n
\end{aligned}$$

This substitution can go upto $n = 0$
Hence we assume that $n - k = 0$
which means $n = k$

$T(n) = T(n-n) + \log(n-n+1) + \log(n-n+2) + \ldots\ldots + \log(n-1) + \log n$
$T(n) = T(0) + \log 1 + \log 2 + \log 3 + \ldots\ldots + \log(n-1) + \log n$
$T(n) = T(0) + \log n!$
$T(n) = 1 + \log n!$

Time Complexity -> Θ(n log n)

Short Cut Method for Recurrence Relation

So far we saw the following Recurrence Relations and their time complexities

1. **$T(n) = T(n-1) + 1 \rightarrow \Theta(n)$**
2. **$T(n) = T(n-1) + n \rightarrow \Theta(n^2)$**
3. **$T(n) = T(n-1) + \log n \rightarrow O(n \log n)$**

   So what do you understand from these three solutions?

   a) In recurrences the function is reducing function every time its called
   b) You are multiplying the **1 with n** for **first solution**, **n with n** in **second solution** and **log n with n** in **third solution**. So to derive the time complexity you are just multiplying right side of the addition operand with n.

Hence we can make out that

4. **$T(n) = T(n-1) + n^2 \rightarrow \Theta(n^3)$**

But what if ?

5. **$T(n) = T(n-2) + 1 \rightarrow \Theta(n)$**
   exact steps would be n/2 but we ignore the constant 2.
6. **$T(n) = T(n-100) + n \rightarrow \Theta(n^2)$**
   note: may be this wont be true for really small value, but time complexity analysis we don't do for small values, we do this only for very big values.

Again what if ?

7. **$T(n) = 2*T(n-1) + 1$**

Recurrence Relation ($T(n) = 2*T(n-1) + 1$)

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

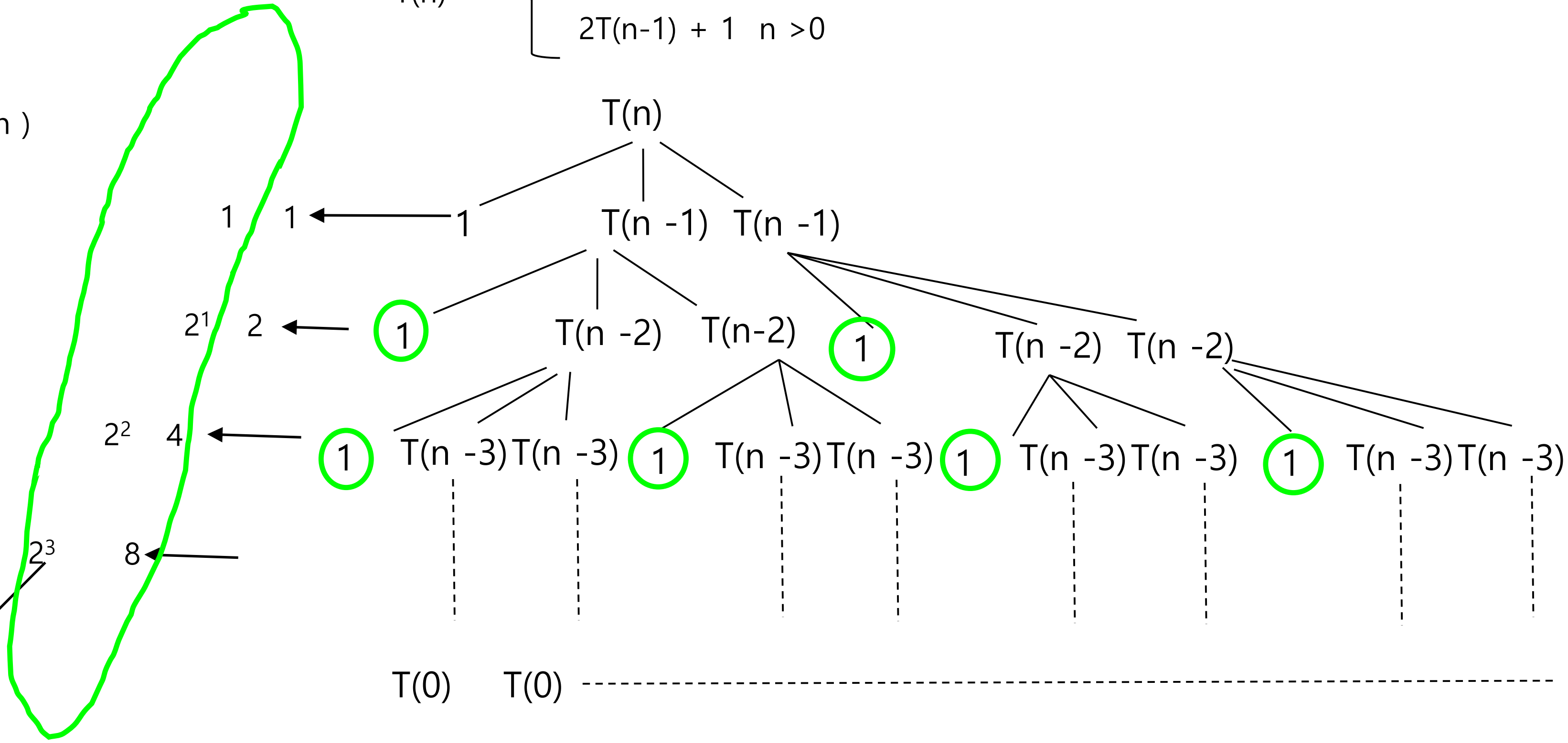
Finding Time complexity using **Recursive Tree**

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

```
Void Test( int n )        ---->  This function takes T( n )
{
   if ( n >0 )
   {

      stmt;               ---->  1
      Test(n-1);          ---->  T ( n -1)
      Test(n-1);          ---->  T ( n - 1)
   }
}
```

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



geometric series

$$1 + 2 + 2^2 + 2^3 + \text{----------} \; 2^k = 2^{k+1} - 1$$

$$a + ar + ar^2 + ar^3 + \text{----------} \; ar^k = a(r^{k+1} - 1)/r-1$$

$a=1,\ r=2$

When $n-k = 0$, i.e $n = k$

Answer : $2^{n+1} - 1 \rightarrow \Theta(2^n)$

Complexity is - $\Theta(2^n)$

Substitution Method (T(n) =2T(n-1) +1)

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

```
Void Test( int n )          → This function takes T( n )
{
   if ( n >1 )
   {

      stmt;                 → 1
      Test(n-1);            → T ( n -1)
      Test(n-1);            → T ( n - 1)
   }
}
```

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

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

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

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

$= 2^3 [2T(n-4) + 1] + 2^2 + 2 + 1 = 2^4 T(n-4) + 2^3 + 2^2 + 2 + 1$

.....

.....

.....

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

This substitution can go upto n = 0
Hence we assume that n – k = 0
which means n = k

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

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

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

Complexity is - $\Theta(2^n)$

Questions : Find Time complexity for the following

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

If possible try to think of a sample code
use recursive and substitution method
Also verify it with the Master's theorem

Master Theorem for Decreasing Function

Masters Theorem for Decreasing function

Note: You can not derive the complexity every time, hence you must memoriz e it.

General Form of recurrence relation

$T(n) = aT(n-b) + f(n)$

$a>0$ $b > 0$ and $f(n) = O(n^k)$ where $k >= 0$

Case 1: If $a = 1$ -> $O(f(n) \times n)$ -> $O(n^{k+1})$ so if f(n) is $O(n^k)$ we just multiply it with one more n.

1. $T(n) = T(n-1) + 1 \rightarrow O(n)$ — $a=1, b=1, f(n) = O(1) \rightarrow O(n)$
2. $T(n) = T(n-1) + n \rightarrow O(n^2)$ — $a=1, b=1, f(n) = O(n) \rightarrow O(n^2)$
3. $T(n) = T(n-1) + \log n \rightarrow O(n \log n)$ — $a=1, b=1, f(n) = O(\log n) \rightarrow O(n \log n)$
4. $T(n) = T(n-1) + n^2 \rightarrow O(n^3)$ — $a=1, b=1, f(n) = O(n^2) \rightarrow O(n^3)$
5. $T(n) = T(n-2) + 1 \rightarrow O(n)$ — $a=1, b=2, f(n) = O(n) \rightarrow O(n)$
6. $T(n) = T(n-1) + n^k \rightarrow O(n^{k+1})$ — $a=1, b=2, f(n) = O(n^k) \rightarrow O(n^{k+1})$

Case 3 : If $a < 1$ -> $O(f(n)$ -> $O(n^k)$

Case 2 : If $a > 1$ -> $O(n^k a^{n/b})$

7. $T(n) = 2*T(n-1) + 1 \rightarrow O(2^n)$ — $a=2, b=1, f(n) = O(1) \rightarrow O(2^n) \rightarrow O(a^n)$
8. $T(n) = 3*T(n-1) + 1 \rightarrow O(3^n)$ — $a=3, b=1, f(n) = O(1) \rightarrow O(3^n) \rightarrow O(a^n)$
9. $T(n) = 2*T(n-1) + n \rightarrow O(n2^n)$ — $a=2, b=1, f(n) = O(n) \rightarrow O(n2^n) \rightarrow O(na^n)$
10. $T(n) = 2*T(n-1) + n^k \rightarrow O(n^k 2^n)$ — $a=2, b=1, f(n) = O(n^k) \rightarrow O(n^k 2^n) \rightarrow O(n^k a^n)$
11. $T(n) = 2*T(n-2) + n^k \rightarrow O(n^k 2^{n/2})$ — $a=2, b=2, f(n) = O(n^k) \rightarrow O(n^k 2^{n/2}) \rightarrow O(n^k a^{n/b})$

Dividing Function

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

Finding Time complexity using **Recursive Tree**

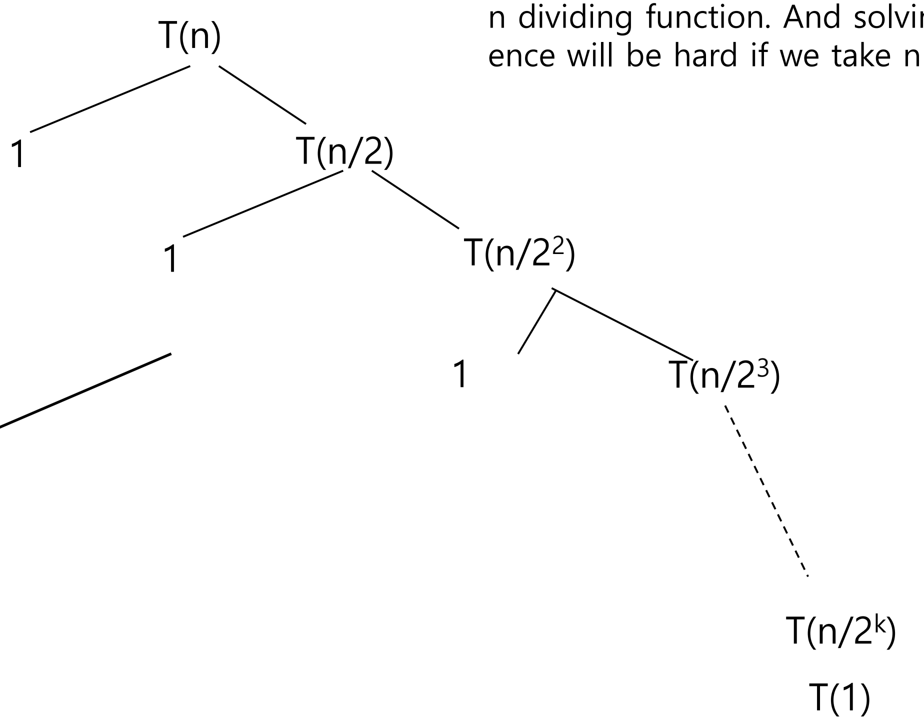
$$T(n) = \begin{cases} 1 & n=1 \\ T(n/2) + 1 & n>1 \end{cases}$$

Note: we are taking the condition as n>1 not 0, as this is dividing function. In decreasing functions we were taking it as 0. There wont be a situation when n=0 in dividing function. And solving a recurrence will be hard if we take n=0

```
Void Test( int n )        ——→ This function takes T( n )
{
   if ( n >1 )
   {
      stmt;               ——→ 1

      Test(n/2);          ——→ T ( n/2)
   }
}
```

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



How many times 1 is added, it added k times. Find the value of k

when will $n/2^k = 1$

$n = 2^k$

$\log n = k \log 2$ — Note : $\log 2 = 1$

$k = \log n$

Ans: Complexity $k \times 1 = \log n \times 1$

Complexity is - $\Theta(\log n)$

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

```
Void Test( int n )      ———→  This function takes T( n )
{
   if ( n >1 )
   {
      stmt;             ———→  1

      Test(n/2);        ———→  T ( n/2)
   }
}
```

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

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

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &= [T(n/2^2) + 1] + 1 = T(n/2^2) + 1 + 1 \\ &= [T(n/2^3) + 1] + 2 = T(n/2^3) + 3 \\ &= [T(n/2^4) + 1] + 3 = T(n/2^4) + 4 \\ &\;\;\ldots\ldots \\ &\;\;\ldots\ldots \\ &\;\;\ldots\ldots \\ &= T(n/2^k) + k \quad (\text{when } n/2^k = 1) \\ &= 1 + \log n \quad (2^k = n,\ T(n/2^k) = T(1) = 1,\ k = \log n) \\ &= \log n \end{aligned}$$

Complexity is - $\Theta(\log n)$

Dividing Function
Recurrence Relation ($T(n) = 2*T(n/2) + n$)

Recursive Method (T(n) =2T(n/2) +n)

Finding Time complexity using **Recursive Tree**

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

Note: we are taking the condition as n>1 not 0, as this is dividing function. In decreasing functions we were taking it as 0. There wont be a situation when n=0 in dividing function. And solving a recurrence will be hard if we take n=0

```
Void Test( int n )        ->  This function takes T( n )
{
   if ( n >1 )
   {
      for(i=0; i< n; i++)
      {
         stmt;            ->  n
      }
      Test(n/2);          ->  T ( n/2)
      Test(n/2);          ->  T ( n/2)
   }
}
```

$\boxed{T(n) = 2 * T(n/2) + n}$

T(n)

n ← n T(n/2) T(n/2)

n ← 2(n/2) ← n/2 T(n/2²) T(n/2²) n/2 T(n/2²) T(n/2²)

n ← 4n/2² ← n/2² T(n/2³) T(n/2³) n/2² T(n/2³) T(n/2³) n/2² T(n/2³) T(n/2³) n/2² T(n/2³) T(n/2³)

n ← 8n/2³ ← n/2³ n/2³ n/2³ n/2³ n/2³ n/2³ n/2³ n/2³

n ← $n/2^k$ $n/2^k$

T(1) T(1)

n + n + n + n + ---------- n = kn

when will $n/2^k = 1$

$n = 2^k$

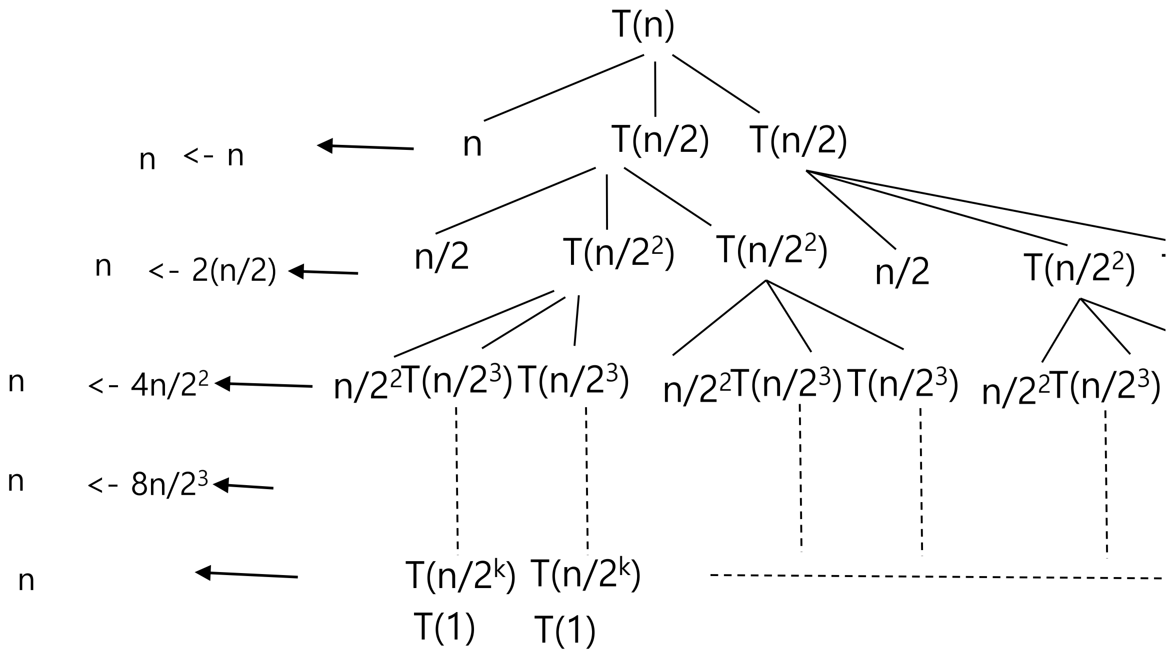
log n = k log2

Note : log 2 = 1

k = log n

Ans: Complexity kn = n log n

Complexity is - Θ(n log n)



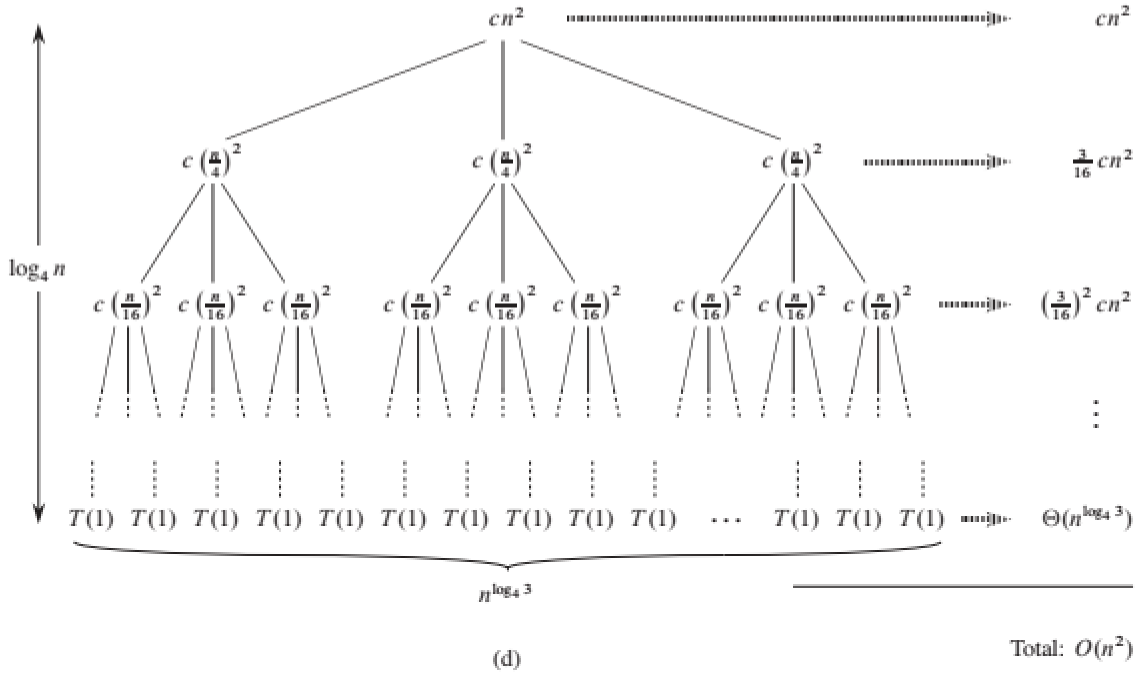


Figure 4.5 Constructing a recursion tree for the recurrence $T(n) = 3T(n/4) + cn^2$. Part **(a)** shows $T(n)$, which progressively expands in **(b)–(d)** to form the recursion tree. The fully expanded tree in part (d) has height $\log_4 n$ (it has $\log_4 n + 1$ levels).

Substitution Method (T(n) =2T(n-1) +n)

```
Void Test( int n )          ──→  This function takes T( n )
{
   if ( n >1 )
   {
      for(i=0; i< n; i++)
      {
         stmt;              ──→  n
      }
      Test(n/2);            ──→  T ( n/2)
      Test(n/2);            ──→  T ( n/2)
   }
}
```

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

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

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

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

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

$= 2^3\,[2T(n/2^4) + n/2^3] + 3n = 2^4\,T(n/2^4) + 3n + n$

.....

.....

.....

$= 2^k\,T(n/2^k) + kn$

$= n \log n \quad (2^k = n\,,\ T(n/2^k) = T(1) = 1,\ k = \log n\ )$

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

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

$T(n) = 2T(n/2) + n$
$T(n/2^3) = 2T((n/2^3)/2) + n/2^3$
$T(n/2^3) = 2T(n/2^4) + n/2^3$

$T(n/2^k) = T(1)$
$n/2^k = 1$
$n = 2^k$
$\log n = k$

Complexity is - Θ (n log n)

Master Theorem for Dividing Function

Masters Theorem for Dividing function Three Cases:

General Form of recurrence relation

$T(n) = aT(n/b) + f(n)$

$a >= 1 \quad b > 1 \quad f(n) = O(n^k \log^p n)$

Case 1: if $\log_b a > k$ then $O(n^{\log_b a})$

Case 2: if $\log_b a = k$

if $p > -1$ $O(n^k \log^{p+1} n)$

if $p = -1$ $O(n^k \log \log n)$

if $p < -1$ $O(n^k)$

Case 3: if $\log_b a < k$

if $p >= 0$ $O(n^k \log^p n)$

if $p < 0$ $O(n^k)$

1) $\log_b a$
2) k

Masters Theorem for Dividing function Case 1:

General Form of recurrence relation

$T(n) = aT(n/b) + f(n)$

$a >= 1 \quad b > 1 \; f(n) = O(n^k \log^p n)$

1) $\log_b a$
2) k

Case 1: if $\log_b a > k$ then $O(n^{\log_b a})$

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

a =2, b=2, f(n) = O(1) -> $O(n^0 \log^0 n)$
what is k and p, k = 0 , p = 0
(p=0 because there is no log n)
What is $\log_b a = \log_2 2$
Case 1: if $\log_b a > k$ then $O(n^{\log_b a})$
Ans: $O(n^1)$

2. $T(n) = 4T(n/2) + n$

a =4, b=2, f(n) = O(n) -> $O(n^1 \log^0 n)$
what is k and p, k = 1 , p = 0
(p=0 because there is no log n)
What is $\log_b a = \log_2 4 = 2$
$\log_b a > k$
Case 1: if $\log_b a > k$ then $O(n^{\log_b a})$
Ans: $O(n^{\log_2 4}) \to O(n^2)$

3. $T(n) = 8T(n/2) + n$

a =8, b=2, f(n) = O(n) -> $O(n^1 \log^0 n)$
what is k and p, k = 1 , p = 0
(p=0 because there is no log n)
What is $\log_b a = \log_2 8 = 3$
$\log_b a > k$
Case 1: if $\log_b a > k$ then $O(n^{\log_b a})$
Ans: $O(n^{\log_2 8}) \to O(n^3)$

4. $T(n) = 8T(n/2) + n^2$

a =8, b=2, f(n) = O(n) -> $O(n^1 \log^0 n)$
what is k and p, k = 2 , p = 0
What is $\log_b a = \log_2 8 = 3$
$\log_b a > k$
Case 1: if $\log_b a > k$ then $O(n^{\log_b a})$
Ans: $O(n^{\log_2 8}) \to O(n^3)$

5. $T(n) = 9T(n/3) + n^2$

a = 9, b=3 , f(n) = $O(n^2) \to O(n^2 \log^0 n)$
what is k and p, k = 2 , p = 0
What is $\log_b a = \log_3 9 = 2$
$\log_b a = k$
This does not come under Case 1-> Case2

Hint: As long as $\log_b a$ is greater than pow er of n, time complexity will be $O(n^{\log_b a})$

Questions : Find Time complexity for the following

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

Use Recursive Tree or Substitution Method

$T(n) = 8T(n/2) + n\log n$

Use Master's theorem

Masters Theorem for Dividing function Case 2:

General Form of recurrence relation

$T(n) = aT(n/b) + f(n)$

$a >= 1 \quad b > 1 \quad f(n) = O(n^k \log^p n)$

Case 2: if $\log_b a = k$

if $p > -1 \quad O(n^k \log^{p+1} n)$

if $p = -1 \quad O(n^k \log \log n)$

if $p <= -1 \quad O(n^k)$

1) $\log_b a$
2) k

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

$a = 2, b=2, f(n) = O(n^1) \rightarrow O(n^1 \log^0 n)$
what is k and p, $k = 1$, $p = 0$
What is $\log_b a = \log_2 2 = 1$
$\log_b a = k$
($p = 0$ because there is no log n)
$p > -1$
Hence based on $O(n^k \log^{p+1} n)$
Ans: $O(n \log n)$

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

$\log_b a = \log_2 4 = 2$
$k = 2$
$\log_b a = k$
Hence based on $O(n^k \log^{p+1} n)$
Ans: $O(n^2 \log n)$

3. $T(n) = 4T(n/2) + n^2 \log n$

$a = 4, b=2, f(n) = O(n^2 \log n) \rightarrow O(n^2 \log^1 n)$
what is k and p, $k = 2$, $p = 1$
What is $\log_b a = \log_2 4 = 2$
$\log_b a = k$
$p > -1$
Hence based on $O(n^2 \log^{p+1} n)$
Ans: $O(n^2 \log^2 n)$

4. $T(n) = 2T(n/2) + n^1/\log n$

$a = 2, b=2, f(n) = O(n^1/\log n) \rightarrow O(n^1 \log^{-1} n)$
what is k and p, $k = 1$, $p = -1$
What is $\log_b a = \log_2 2 = 1$
$\log_b a = k$
$p = -1$
Hence based on $O(n^k \log \log n)$
Ans: $O(n \log \log n)$

Hint: Take this as it is and multiply it by log n

General Form of recurrence relation

$T(n) = aT(n/b) + f(n)$

$a >= 1 \quad b > 1 \quad f(n) = O(n^k \log^p n)$

Case 2: if $\log_b a = k$

if $p > -1$ $O(n^k \log^{p+1} n)$

if $p = -1$ $O(n^k \log \log n)$

if $p <= -1$ $O(n^k)$

1) $\log_b a$
2) k

Case 1: if $\log_b a > k$ then $O(n^{\log_b a})$

1. $T(n) = 2T(n/2) + n^1 / \log^2 n$

$a = 2$, $b=2$, $f(n) = O(n^1/\log^2 n) \rightarrow O(n^1 \log^{-2} n)$
what is k and p, $k = 1$, $p = -2$
What is $\log_b a = \log_2 2 = 1$
$\log_b a = k$
$p <= -1$
Hence based on $O(n^k)$
Ans: $O(n)$

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

$a = 1$, $b=2$, $f(n) = O(n^2) \rightarrow O(n^2 \log^0 n)$
what is k and p, $k = 2$, $p = 0$
What is $\log_b a = \log_2 1 = 0$
$\log_b a < k$

This doesn't satisfy Case 2, neither it satisfies Case 1 -> Case 3

Masters Theorem for Dividing function Case 3:

General Form of recurrence relation

$T(n) = aT(n/b) + f(n)$

$a >= 1$ $b > 1$ $f(n) = O(n^k \log^p n)$

Case 3: if $\log_b a < k$

if $p >= 0$ $O(n^k \log^p n)$

if $p < 0$ $O(n^k)$

1) $\log_b a$
2) k

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

$a = 1, b=2$, $f(n) = O(n^2) \rightarrow O(n^2 \log^0 n)$
what is k and p, $k = 2$, $p = 0$
What is $\log_b a = \log_2 1 = 0$
$\log_b a < k$
$p>=0$
Based on $O(n^k \log^p n)$
Ans: $O(n^2)$

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

$k = 2$
$\log_b a = \log_2 2 = 1$
$\log_b a < k$
$p>=0$
Based on $O(n^k \log^p n)$
Ans: $O(n^2)$

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

$k = 2, p = 2$
$\log_b a = \log_2 2 = 1$
$\log_b a < k$
$p=2$
$P>=0$
Based on $O(n^k \log^p n)$
Ans: $O(n^2 \log^2 n)$

Hint: Take directly f(n)

Masters Theorem for Dividing function All Cases

Note: You can not derive the complexity every time, hence you must memorize it.

General Form of recurrence relation

$T(n) = aT(n/b) + f(n)$

$a >= 1 \quad b > 1 \quad f(n) = O(n^k \log^p n)$

1) $\log_b a$
2) k

Case 1: if $\log_b a > k$ then $O(n^{\log_b a})$

Case 2: if $\log_b a = k$

- if $p > -1$ $O(n^k \log^{p+1} n)$
- if $p = -1$ $O(n^k \log \log n)$
- if $p < -1$ $O(n^k)$

Case 3: if $\log_b a < k$

- if $p >= 0$ $O(n^k \log^p n)$
- if $p < 0$ $O(n^k)$

Questions : Find Time complexity for the following

$T(n) = 4T(n/2) + n\log^5 n$

$T(n) = 9T(n/3) + n^2$

$T(n) = 8T(n/2) + n^3$

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

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

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

$T(n) = \sqrt{n}\,T(\sqrt{n}) + n$

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

Use Master's theorem

Substitution Method ($T(n) = T(\sqrt{n}) + 1$)

Note: we are taking the condition as n>2 not 0 or 1. There wont be a situatio n when n=0 or 1 in these function. And solving a recurrence will be hard if we ta ke n=0 or n=1. If you have root function, smallest value should be grater than or e qual to 2

```
Void Test( int n )        → This function takes T( n )
{
   if ( n >2 )
   {
      stmt;              → 1
      Test(√n);          → T (√n)
}
}
```

$\boxed{T(n) = T(\sqrt{n}) + 1}$

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

$T(n) = T(\sqrt{n}) + 1$
$T(n) = T(n^{1/2}) + 1$
$= [\, T(n^{1/4}) + 1 \,] + 1 = T(n^{1/4}) + 2$
$= [\, T(n^{1/8}) + 1 \,] + 2 = T(n^{1/8}) + 3$
.....
.....
.....
$= T(n^{1/2^k}) + k$ → Eq. 1

As per the base case assumption

$n^{1/2^k} = 2$

Take log on both the sides

$(\frac{1}{2})^k \log_2 n = 1$

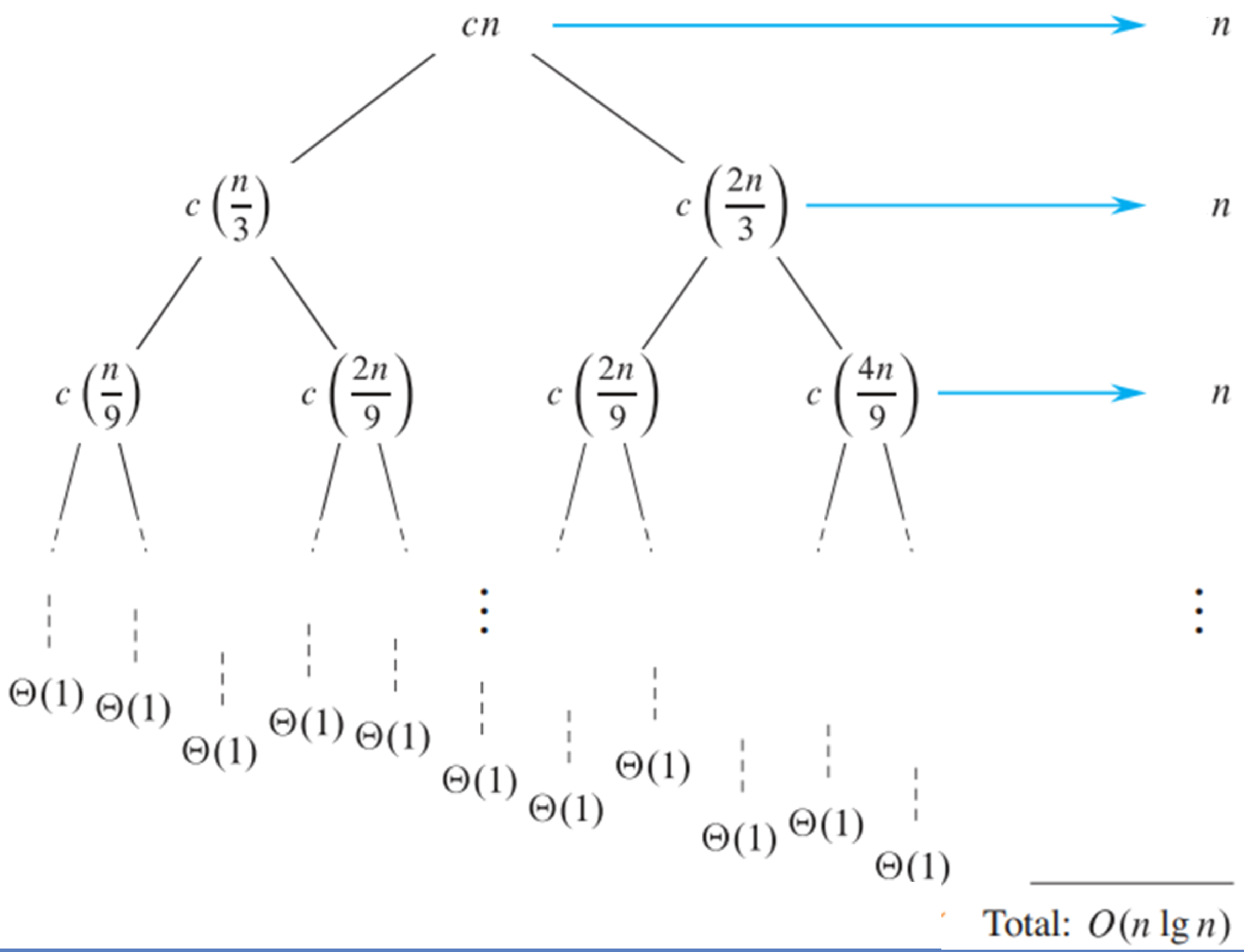
$\log_2 n = 2^k$

Again take the log

$\log \log_2 n = k$

Complexity is - $\Theta(\log \log_2 n)$

An irregular example : T(n) = T(n/3) + T(2n/3) + n



$$T(n) = \begin{cases} 1 & n=1 \\ T(n/3) + T(2n/3) + n & n>1 \end{cases}$$

This will go upto **k^th** iteration

At **k^th** iteration

$(2/3)^k n = 1$

$n = (3/2)^k$

$\log_{3/2} n = k \log_{3/2} 3/2$

$k = \log_{3/2} n$

add all these steps k time

$O(n \log_{3/2} n)$