Analysis of Recursion

```
def fun(n):
        if n == 1:
                return
        for i in range(n):
                print("GFG")
        fun(n/2)
        fun(n/2)
```

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

$T(1) = \Theta(1)$

```
def fun(n):
        if n == 1:
                return
        print("GFG")
        fun(n/2)
        fun(n/2)
```

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

$T(1) = \Theta(1)$

```
def fun(n):
        if n == 1:
                return
        print(n)
        fun(n - 1)
```

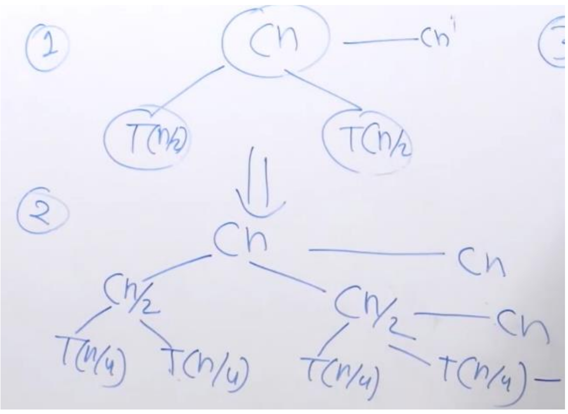
$T(n) = T(n - 1) + \Theta(1), n > 1$

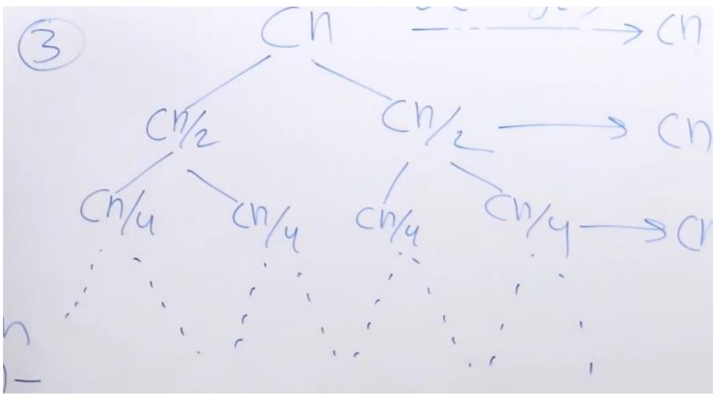
$T(1) = \Theta(1)$

<u>Recursion Tree method</u>

- We write non-recursive part as root of tree and recursive parts as children.
- We keep expanding children until we see a pattern.

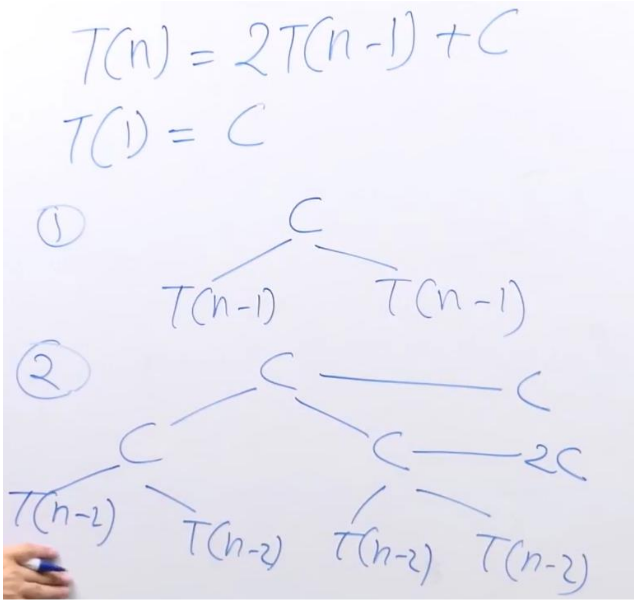
$T(n) = 2T(n/2) + Cn$	$Cn + Cn + Cn + \ldots\ldots + Cn$
$T(1) = C$	$\Rightarrow \log_2 n$
	⇨ $Cn \times \log_2 n$
	⇨ $\Theta(n \log n)$

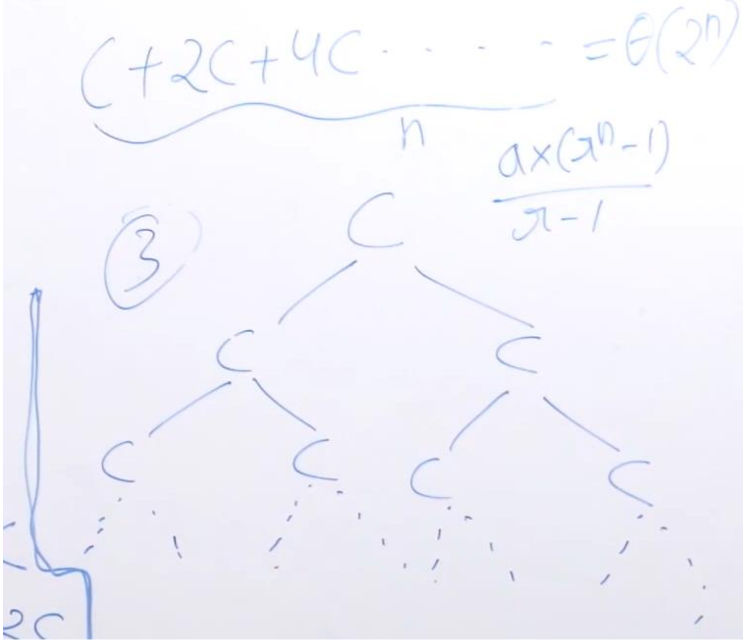




$T(n) = T(n - 1) + C$

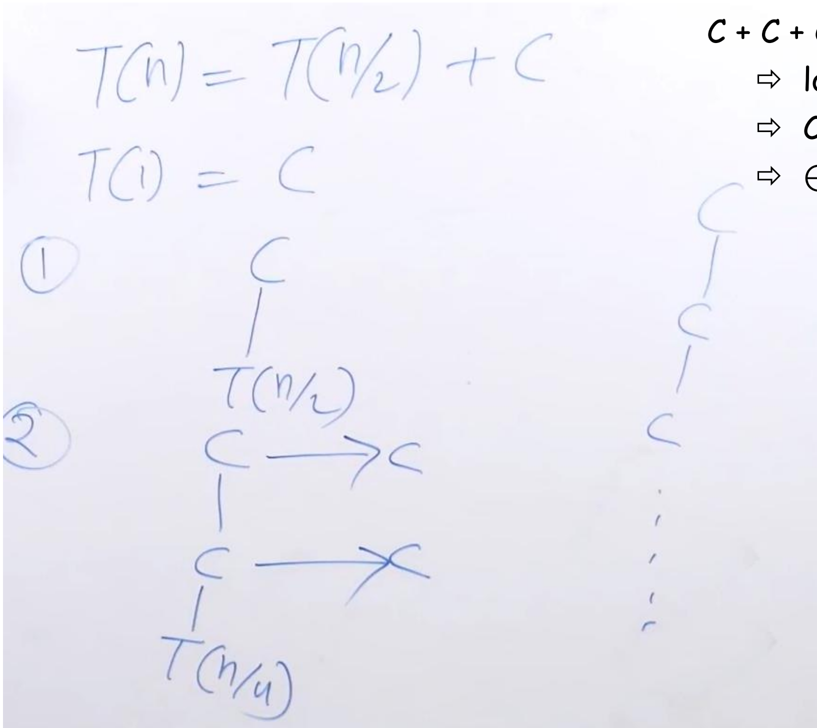
$T(1) = C$





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

$T(1) = C$



$C + C + C + C$ …………

- ⇨ $\log_2 n$
- ⇨ $C \times \log_2 n$
- ⇨ $\Theta(\log n)$

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

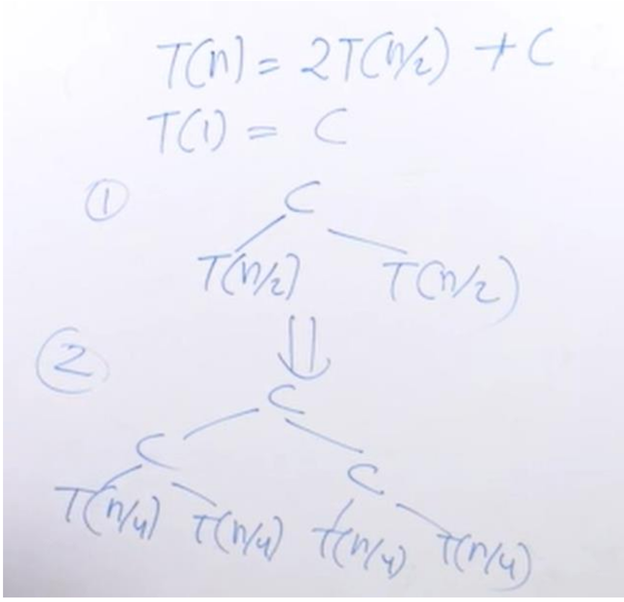
$T(1) = C$

$C + 2C + 4C + \ldots\ldots\ldots\ldots$

⇨ $\log_2 n$

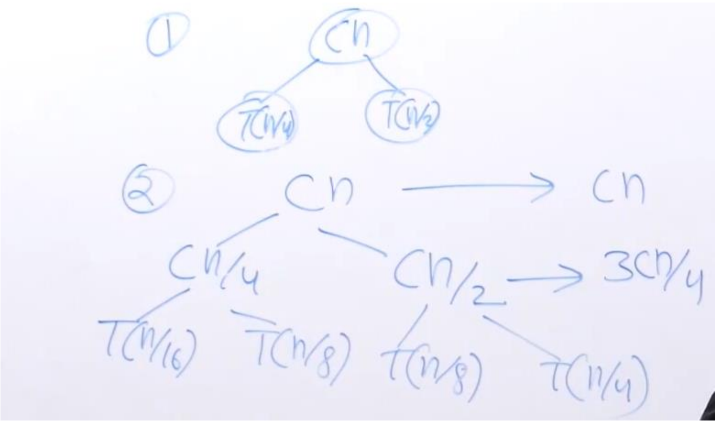
⇨ $\frac{C \times (2^{\log_2 n} - 1)}{2-1}$

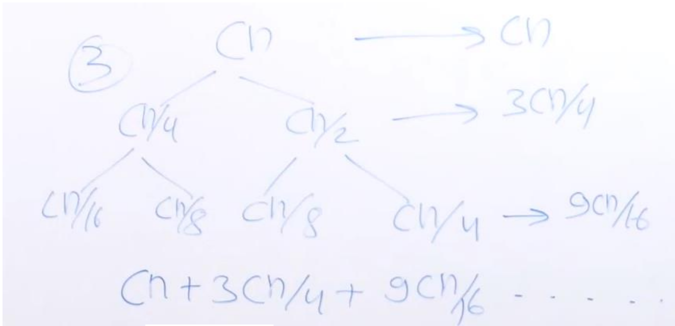
⇨ $\Theta(n)$



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

$T(1) = C$



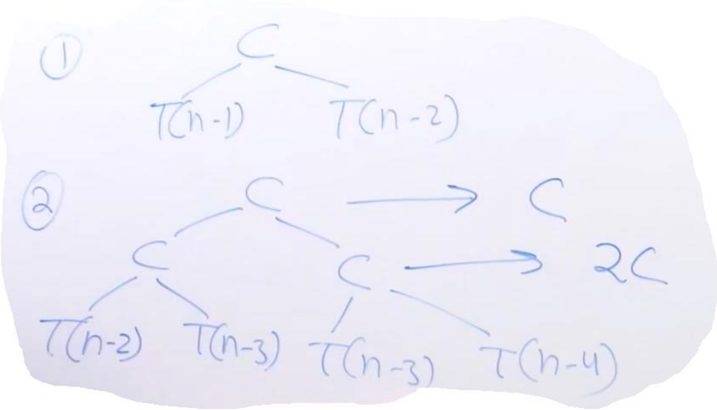


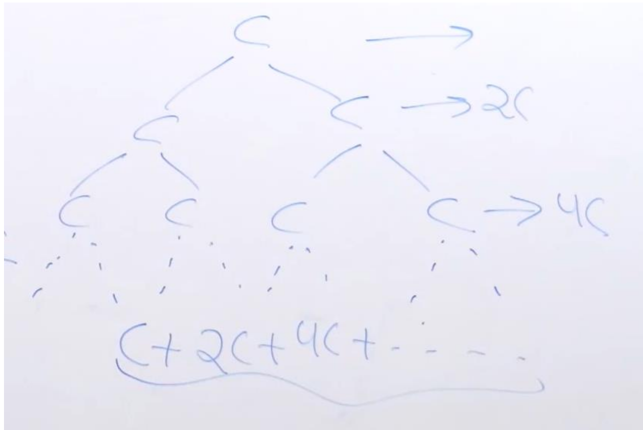
log(n)

$O(Cn \times 1/(1-3/4)) \Rightarrow O(n)$

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

$T(1) = C$





$C(1 + 2 + 4 + \ldots\ldots + n \text{ terms}) \Rightarrow O(2^n)$