

## → Big Notion $\Theta$ :

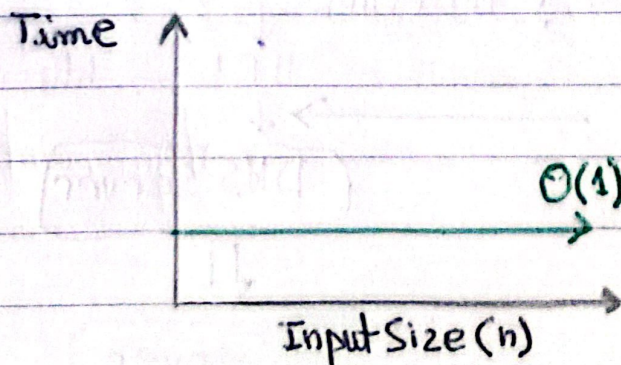
### → What is Big $\Theta$ ?

\* est un outil mathématique utilisé en info pour décrire la complexité d'un algorithme, indiquant comment le temps ou l'espace mémoire nécessaire évolue en fct d'inputs size.

### → $O(1)$ :

\* La complexité constante, signifie que le temps d'exécution ou l'espace mémoire requis par un algo est fixe.

\* Autrement dit, l'algorithme s'exécute en temps constant, indépendamment de la quantité de donnée à traiter.



### Exemple (Python)

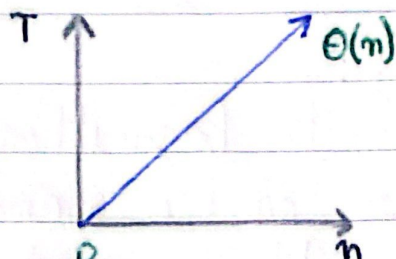
```
arr = [1, 2, 3]
arr.append(4)
arr.pop()
arr[0]
```



→  $\Theta(n)$  :

\* désigne une complexité linéaire, indiquant que le temps d'exécution ou l'espace mémoire nécessaire pour un algorithme augmente de façon proportionnelle à la taille d'input.

$n \uparrow \Leftrightarrow T \uparrow$



$$\Theta(2n) = \Theta(n)$$

Exemple:

→ `arr = [1, 2, 3]`

`for i in arr:`

`print(i)`

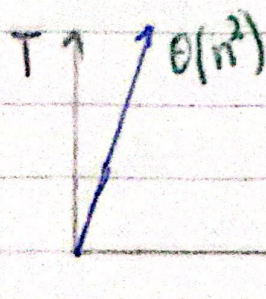
→ `arr2 = [1, 2, 3, 4, 5]`

`sum(arr2)`

→  $\Theta(n^2)$  :

\* désigne une complexité quadratique, où le temps d'exécution ou l'espace mémoire pour un algo croît proportionnellement au carré de la taille d'input.

\* Par exemple, si la taille d'input ( $n=2$ ),  
Temps || l'espace = 4



$$\Theta(n^2 + n) = \Theta(n^2)$$

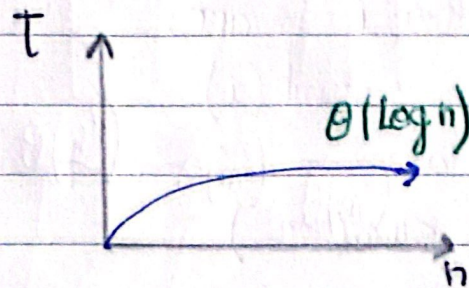


Example:

```
def print_elements(n):  
    for i in range(n):  
        for j in range(n):  
            print(i, j)  
res = print_elements(4)
```

→  $\Theta(\log(n))$ :

\* complexité logarithmique. signifie que le t d'exécution ou l'es mémoire requis par l'algo augmente logarithmiquement par rapport à taille d'input.

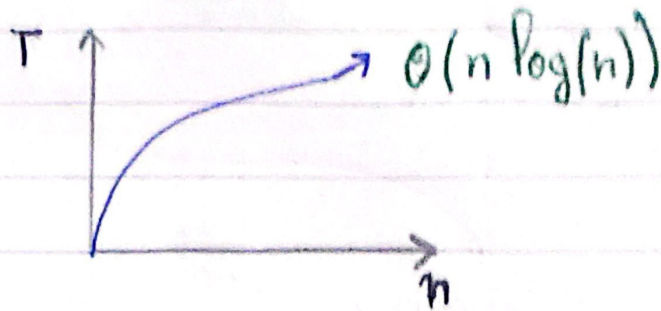


Example:

```
def recherche_binaire(arr, el):  
    debut = 0  
    fin = len(arr) - 1  
    while debut <= fin:  
        milieu = (debut + fin) // 2  
        if arr[milieu] == el:  
            return milieu  
        elif arr[milieu] < el:  
            debut = milieu + 1  
        else:  
            fin = milieu - 1  
    return -1
```



→  $\Theta(n \cdot \log(n))$ :

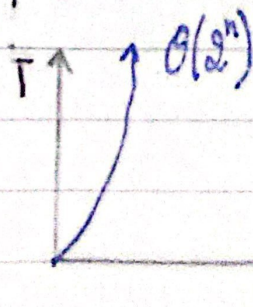


Example:

```
import heapq
arr = [1, 2, 3]
heapq.heapij(arr) #  $\Theta(n)$ 
while arr:
    heapq.heappop(arr) #  $\log(n)$ 
```

→  $\Theta(2^n) : c^n$

\* Pour chaque augmentation de 1 de taille d'input le temps d'exécution ou l'espace nécessaire double.



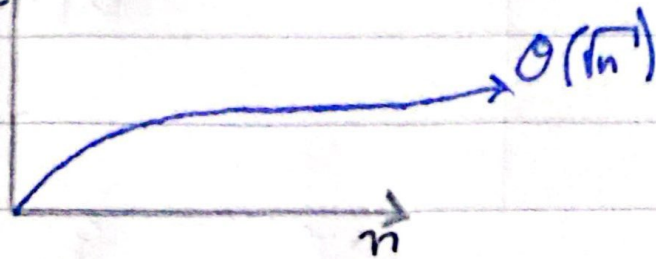
Example:

```
def Fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return Fibonacci(n-1) + Fibonacci(n-2)
```



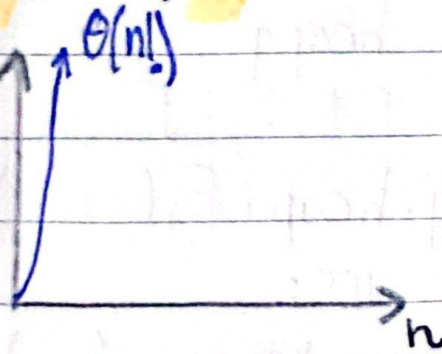
→  $\theta(\sqrt{n})$ :

Time



→  $\theta(n!)$

Time



RECAP:

