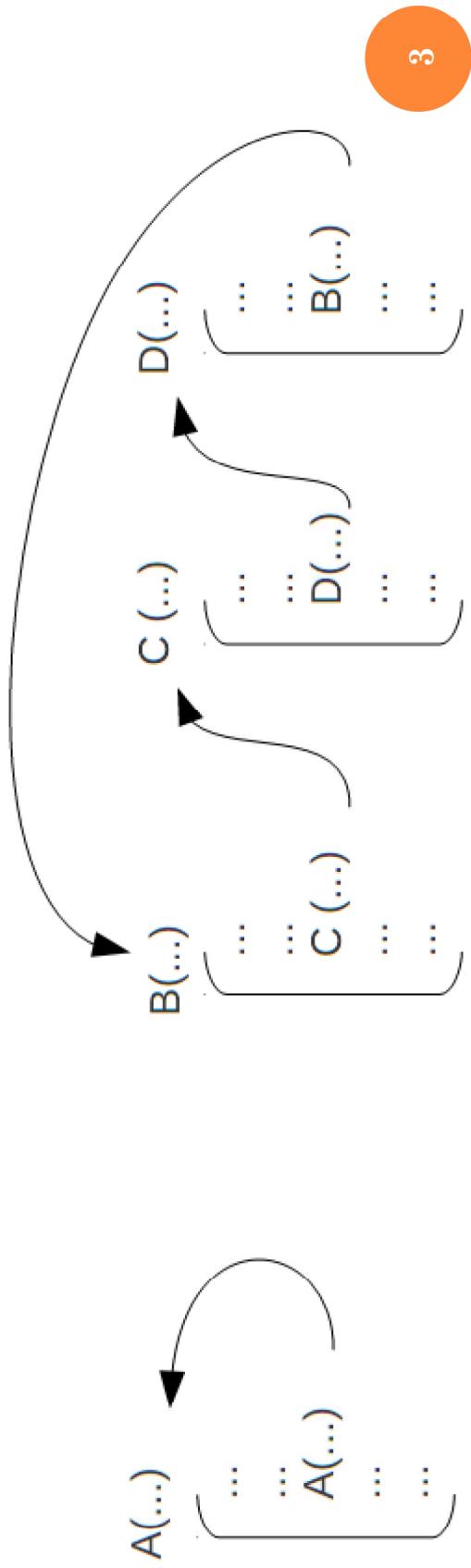


# PLAN DU CHAPITRE IV

- Définition
- Évolution d'un appel récursif
- Types de la récursivité
- Paradigme « Diviser pour régner »
- Conception d'un algorithme récursif
- Complexité des algorithmes récursifs

# DÉFINITION

- ❖ Un algorithme est dit récursif lorsqu'il est **auto-référent**: il fait référence à lui-même dans sa définition.
- ❖ Autrement dit, un algorithme est dit récursif s'il est défini en fonction de lui-même, directement ou indirectement.



3

Récurseur directe

Récurseur indirecte

# EXAMPLE

- ❖ Exemple : la factorielle n!

Itératif	Récuratif
$n! = 1 * 2 * \dots * n$	$n! = n * (n-1)!$

```
Facto (n: entier): entier
Début
    F← 1;
    Pour i← 2 à n faire
        F← F*i;
    retourner F;
Fin
```

# ÉVOLUTION D'UN APPEL RÉCURSIF

- ❖ L'exécution d'un appel récursif passe par deux phases, la phase de descente et la phase de la remontée :
  - Dans la phase de descente, chaque appel récursif fait à son tour un appel récursif.

```
Facto (n: entier): entier  
Début  
    retourner n*Facto (n-1);  
Fin
```

Facto(4)  $\leftarrow$  4 \* Facto(3)  
Facto(3)  $\leftarrow$  3 \* Facto(2)  
Facto(2)  $\leftarrow$  2 \* Facto(1)  
Facto(1)  $\leftarrow$  1 \* Facto(0)

*Phase de descente*

Facto(4)  $\leftarrow$  0 \* Facto(-1)

# ÉVOLUTION D'UN APPEL RÉCURSIF

## CONDITION D'ARRÊT

- ❖ Puisqu'un algorithme récursif s'appelle lui-même, il est impératif qu'on prévoit une condition d'arrêt à la récursion, sinon le programme ne s'arrête jamais!

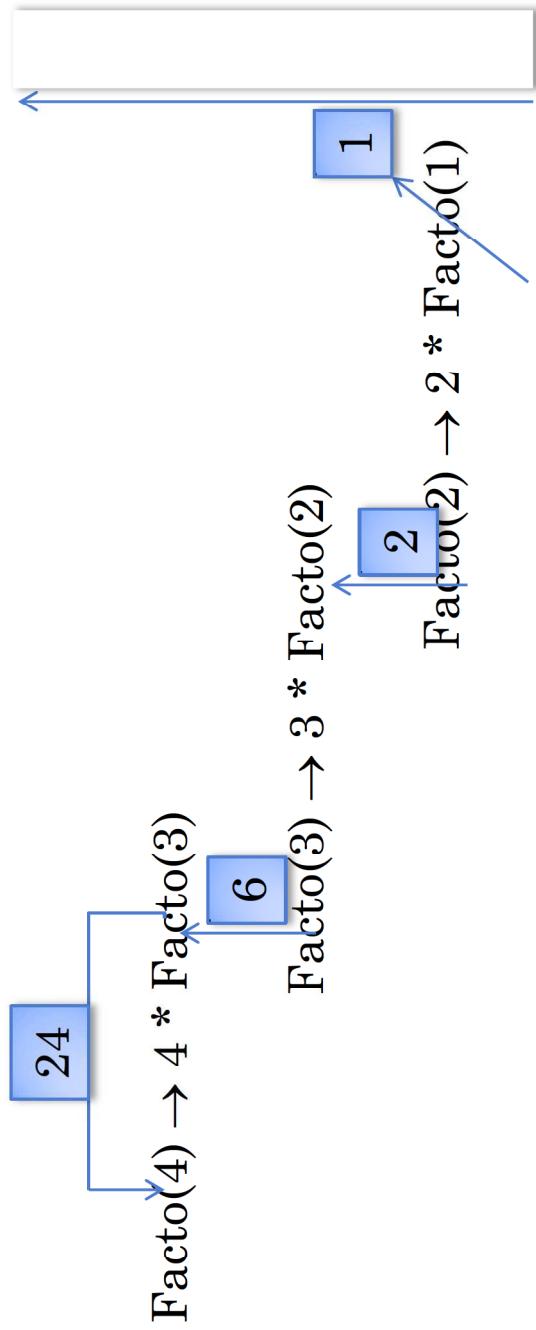
- ❖ **Exemple 1:** la factorielle n!

Version précédente	Version correcte
Facto (n: entier): entier Début retourner n*Facto (n-1); Fin	Facto (n: entier): entier Début <b>Si (n=1) alors retourner 1;</b> Sinon retourner n*Facto (n-1); Fin

# ÉVOLUTION D'UN APPEL RÉCURSIF

- En arrivant à la condition terminale, on commence la **phase de la remontée** qui se poursuit jusqu'à ce que l'appel initial soit terminé, ce qui termine le processus récursif.

7



# EXERCICE 1

1. **Som (n)** calcule la somme des n premiers naturels
2. **Som (a, b)** calcule la somme des entiers a et b
3. **Prod (a, b)** calcule le produit de entiers a et b
4. **Puiss (a, b)** calcule a puissance b
5. **Quotient (a, b)** calcule le quotient de a par b
6. **Reste (a, b)** calcule le reste de division de a par b
7. **PGCD (a, b)** calcule le Plus Grand Commun Diviseur entre a et b
8. **Fib (n)** calcule la suite de Fibonacci ( $U_n = U_{n-1} + U_{n-2}$  si  $n > 1$ , sinon  $U_0 = U_1 = 1$ )

# EXERCICE 1

1. La somme des n premiers

Itératif	Récuratif
$\text{Som}(n) = 0 + 1 + 2 + \dots + n$	$\text{Som}(n) = n + \text{Som}(n-1)$

```
Som(n: entier): entier
Début
    Si n = 0 alors
        retourner (0)
    Sinon
        Pour i <= 1 à n faire
            S <- S+i;
        retourner S;
    Fin
Fin
```

```
Som(n: entier): entier
Début
    Si n = 0 alors
        retourner (0)
    Sinon
        retourner (Som(n-1) +n)
    Fin
Fin
```

# EXERCICE 1

2. La somme de deux entiers a et b

	<b>1ère solution</b>	<b>2ème solution</b>	
	$a+b = a+(b-1)+1$	$a+b = (a-1)+b+1$	
Som (a,b: entier): entier		Som (a,b: entier): entier	
Début		Début	
Si b = 0 alors		Si a = 0 alors	
retourner (b)		retourner (a)	
Sinon		Sinon	
retourner (Som(a, b-1) +1)		retourner (Som(a-1, b) +1)	
Fin		Fin	

10

# EXERCICE 1

3. Le produit de deux entiers a et b

1ère solution	2ème solution	
$a*b = a*(b-1)+a$	$a*b = (a-1)*b+b$	
Prod (a,b: entier): entier	Prod (a,b: entier): entier	
Début	Début	
Si a = 0 ou b = 0 alors	Si a = 0 ou b = 0 alors	
retourner (0)	retourner (0)	
Sinon	Sinon	
retourner (Prod(a, b-1) +a)	retourner (Prod(a-1, b) +b)	
Fin	Fin	

# EXERCICE 1

4. La puissance  $a^b$  (a et b deux entiers positifs)

itératif	Récuratif
$a^b = a * a * a * ..... * a, b \text{ fois}$	<p>Puiss (a,b: entier): entier</p> <p>Début</p> <p>Si b = 0 alors</p> <p style="padding-left: 20px;">retourner (1)</p> <p>Si non</p> <p style="padding-left: 20px;">retourner (Puiss(a, b-1) * a)</p> <p>Fin</p>

# EXERCICE 1

5. Le quotient et le reste de la division de a par b (a et b deux entiers positifs)

Quotient $a \text{ div } b = (a-b) \text{ div } b + 1$	Reste de la division $a \bmod b = (a-b) \bmod b$	Reste (a,b: entier): entier	Début	Si $a < b$ alors retourner (0)	Reste (a,b: entier): entier	Début	Si $a < b$ alors retourner (a)	Reste (a,b: entier): entier	Fin
Quotient(a,b: entier): entier			Début	Si $a < b$ alors retourner (0)		Fin		retourner (Reste(a-b, b))	

# EXERCICE 1

7. Le Plus Grand Commun Diviseur entre a et b (a et b deux entiers positifs)

$$PGCD(a, b) = \begin{cases} PGCD(a - b, b) & \text{si } a > b \\ PGCD(a, b - a) & \text{si } a < b \\ a & \text{si } a = b \end{cases}$$

PGCD(a,b: entier): entier

Début

Si a=b alors retourner (a)

Sinon

Si a < b alors retourner (PGCD(a, b-a))

Sinon retourner (PGCD(a-b, b))

Fin

# EXERCICE 1

## 8. La suite de Fibonacci

$$Fib(n) = \begin{cases} 1 & \text{si } n = 0 \text{ ou } n = 1 \\ Fib(n-1) + Fib(n-2) & \text{sinon} \end{cases}$$

Fib(n: entier): entier

Début

Si n = 0 ou n = 1 alors retourner (1)

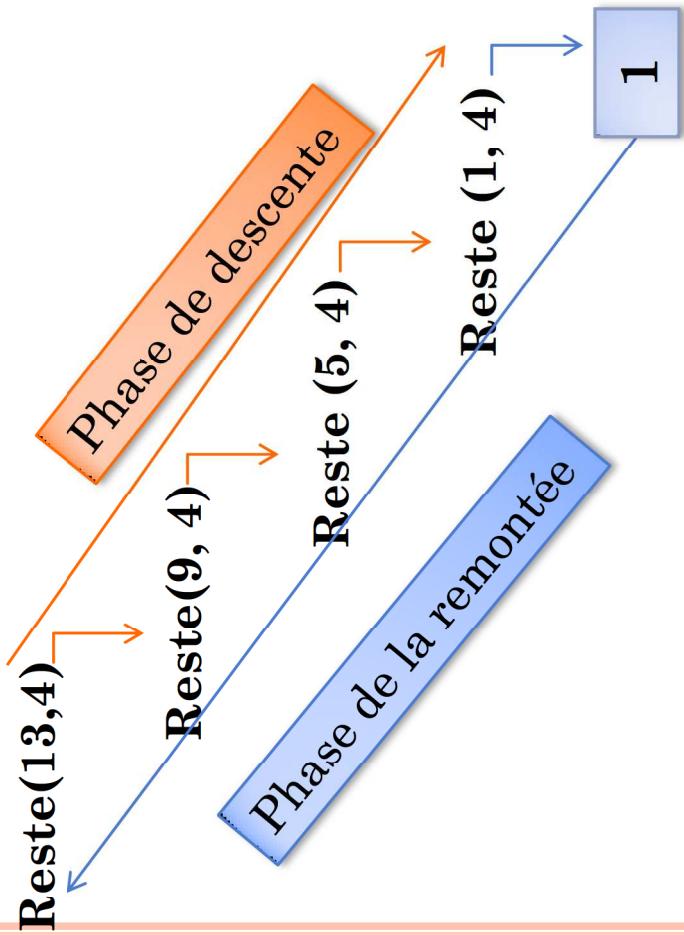
Sinon

    retourner (Fib(n-1)+Fib(n-2))

Fin

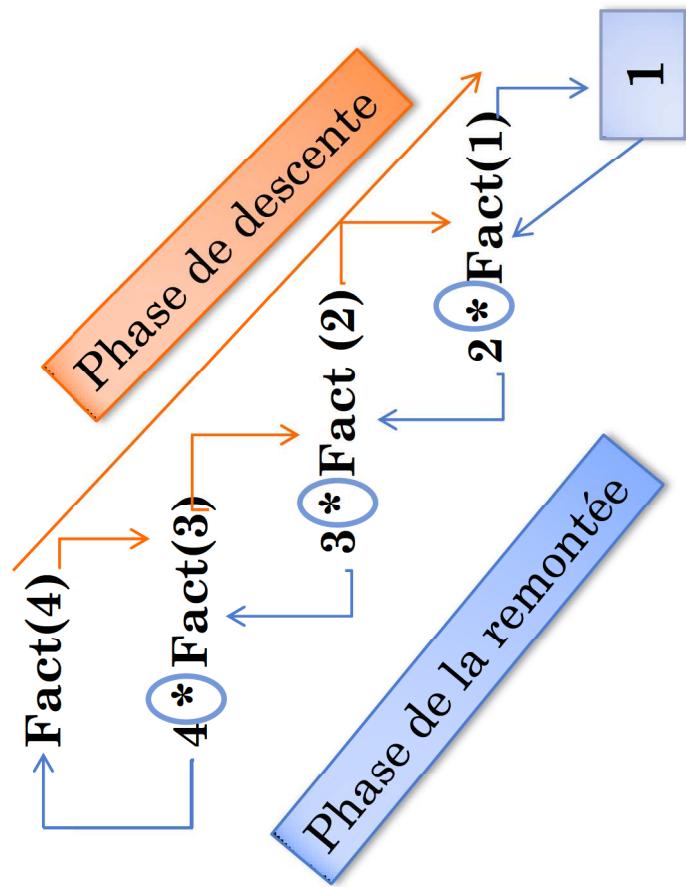
# RÉCURSIVITÉ TERMINALE VS NON TERMINALE

- ❖ Un module récursif est dit terminal si aucun traitement n'est effectué à la remontée d'un appel récursif (sauf le retour du module).



# RÉCURSIVITÉ TERMINALE VS NON TERMINALE

- ❖ Un module récursif est dit **non terminal** si le résultat de l'appel récursif est utilisé pour réaliser un traitement (en plus du retour du module).



# RÉCURSIVITÉ TERMINALE VS NON TERMINALE

Exercice 1	Algorithme	Type
Som (n)	Si $n = 0$ alors retourner (0) Sinon retourner ( $\text{Som}(n-1) + n$ )	Non terminale
Som (a,b)	Si $b = 0$ alors retourner (a) Sinon retourner ( $\text{Som}(a, b-1) + 1$ )	Non terminale
Prod (a,b)	Si $a = 0$ ou $b = 0$ alors retourner (0) Sinon retourner ( $\text{Prod}(a, b-1) + a$ )	Non terminal e
Puissa (a, b)	Si $b = 0$ alors retourner (1) Sinon retourner ( $\text{Puiss}(a, b-1) * a$ )	Non terminale
Quotient (a,b)	Si $a < b$ alors retourner (0) Sinon retourner ( $\text{Quotient}(a-b, b) + 1$ )	Non terminale

# RÉCURSIVITÉ TERMINALE VS NON TERMINALE

Exercice 1	Algorithme	Type
PCGD (a, b)	Si $a=b$ alors retourner (a) Sinon Si $a < b$ alors retourner (PGCD(a, b-a)) Sinon   retourner (PGCD(a-b, b))	Terminale
Fib (n)	Si $n = 0$ ou $n = 1$ alors retourner (1) Sinon retourner (Fib (n-1)+Fib(n-2))	Non terminale
Comb (p, n)	Si $p = 0$ ou $p = n$ alors retourner (1) Sinon retourner (Comb(p,n-1)+Comb(p-1, n-1))	Non terminale

# RÉCURSIVITÉ TERMINALE VS NON TERMINALE

- ❖ **PARFOIS**, il est possible de transformer un module récursif non terminal à un module récursif terminal.

Récusivité non terminale	Récusivité Terminale
Fonction Facto (n: entier): entier Début Si (n=0) alors retourner 1 Sinon retourner n*Facto (n-1); Fin	resultat ←1 <b>Procédure Facto (</b> <b>n: entier, Var résultat: entier)</b> Début Si (n > 1) alors <b>Facto (n-1, n * résultat);</b> Fin

# RÉCURSIVITÉ TERMINALE VS NON TERMINALE

```
resultat ← 1
Procédure Facto ( n: entier, Var resultat: entier)
Début
Si (n > 1) alors
Facto (n-1, n * resultat);
Fin
```

Phase de descente

Facto(4,1) ← Facto(3, 4\*1) ← Facto(2, 3\*4) ← Facto(1, 2\*12) ← Facto(1, **24**)

Phase de la remontée

# RÉCURSIVITÉ TERMINALE VS NON TERMINALE

Fonction non terminale	Procédure terminale
<b>Som (n:entier):entier</b> Si n = 0 alors retourner (0) Sinon retourner (Som(n-1) +n)	<b>resultat←0</b> <b>Som (n: entier, var resultat:entier)</b> Si n > 0 alors Som(n-1, resultat +n))
<b>Som (a,b:entier):entier</b> Si b = 0 alors retourner (a) Sinon retourner ( Som(a, b-1) +1)	<b>resultat←a</b> <b>Som (a,b: entier, var resultat:entier)</b> Si b > 0 alors Som(a, b-1, resultat+1)
<b>Prod (a,b:entier):entier</b> Si a = 0 ou b = 0 alors retourner (0) Sinon retourner (Prod(a, b-1) +a)	<b>resultat←0</b> <b>Prod ( a,b;; entier, var resultat:entier)</b> Si a ≠ 0 et b ≠ 0 alors Prod(a, b-1, resultat +a))

# RÉCURSIVITÉ TERMINALE VS NON TERMINALE

Fonction non terminale	Procédure terminale
<b>Puissance ( a, b:entier):entier</b> Si $b = 0$ alors retourner (1) Sinon retourner (Puiss(a, b-1) *a)	<b>resultat←1</b> <b>Puissance ( a, b: entier, var resultat:entier)</b> Si $b \neq 0$ alors Puiss(a, b-1, resultat *a))
<b>Quotient( a,b:entier):entier</b> Si $a < b$ alors retourner (0) Sinon retourner (Quotient(a-b, b) +1))	<b>resultat←0</b> <b>Quotient ( a,b: entier, var resultat:entier)</b> Si $a \geq b$ alors Quotient(a-b, b, resultat +1))
<b>Fib (n:entier): entier</b> Si $n = 0$ ou $n = 1$ alors retourner (1) Sinon retourner ( Fib (n-1)+Fib(n-2))	<b>IMPOSSIBLE</b>
<b>Comb (p, n: entier): entier</b> Si $p = 0$ ou $p = n$ alors retourner (1) Sinon retourner (Comb(p,n-1)+Comb(p-1, n-1))	

# TYPES DE RÉCURSIVITÉ

1. La récursivité simple où l'algorithme contient un seul appel récursif dans son corps.

Exemple : la fonction factorielle

Récursivité non terminale	Récursivité Terminale
Facto (n: entier): entier Début Si (n=0) alors retourne 1 Sinon retourne n*Facto (n-1); Fin	resultat $\leftarrow$ 1 Procédure Facto ( n: entier, Var resultat: entier) Début Si (n > 1) alors Facto (n-1, n * resultat); Fin

# TYPES DE RÉCURSIVITÉ

- 2. La récursivité multiple où l'algorithme contient plus d'un appel récursif dans son corps

- Exemple: le calcul de la suite de la suite de Fibonacci

$$Fib(n) = \begin{cases} 1 & \text{si } n = 0 \text{ ou } n = 1 \\ Fib(n-1) + Fib(n-2) & \text{sinon} \end{cases}$$

Fib(n: entier): entier

Si n = 0 ou n = 1 alors retourner (1)

Sinon

retourner (Fib (n-1)+Fib(n-2))

# TYPES DE RÉCURSIVITÉ

Exercice 1	Algorithme	Type
Som (n)	Si $n = 0$ alors retourner (0) Sinon retourner (Som(n-1) +n)	Simple
Som (a,b)	Si $b = 0$ alors retourner (a) Sinon retourner (Som(a, b-1) +1)	Simple
Prod (a,b)	Si $a = 0$ ou $b = 0$ alors retourner (0) Sinon retourner (Prod(a, b-1) *a)	Simple
Quotient (a,b)	Si $a < b$ alors retourner (0) Sinon retourner (Quotient(a-b, b) +1)	Simple
Reste (a, b)	Si $a < b$ alors retourner (a) Sinon retourner (Reste(a-b, b))	Simple
Puissance (a, b)	Si $b = 0$ alors retourner (1) Sinon retourner (Puiss(a, b-1) *a)	Simple
PGCD (a, b)	Si $a=b$ alors retourner (a) Sinon Si $a < b$ alors retourner (PGCD(a, b-a)) Sinon retourner (PGCD(a-b, b))	Multiple

# TYPES DE RÉCURSIVITÉ

- 3. La récursivité mutuelle: Des modules sont dits mutuellement récursifs s'ils dépendent les unes des autres
  - Exemple 10 : la définition de la parité d'un entier peut être écrite de la manière suivante :

$$\text{pair}(n) = \begin{cases} \text{vrai} & \text{si } n = 0; \\ \text{impair}(n - 1) & \text{sinon;} \end{cases} \quad \text{et} \quad \text{impair}(n) = \begin{cases} \text{faux} & \text{si } n = 0; \\ \text{pair}(n - 1) & \text{sinon.} \end{cases}$$

Fonction Pair( $n$  : entier) : Boolean

Si ( $n = 0$ ) Alors	Fonction Impair( $n$ : entier) : Boolean
Retourner vrai;	Si ( $n = 0$ ) Alors
Fin Si	Retourner faux;
Retourner Impair( $n - 1$ );	Fin Si
	Fin

# TYPES DE RÉCURSIVITÉ

- 4. La récursivité imbriquée consiste à faire un appel récursif à l'intérieur d'un autre appel récursif.

► Exemple 11: La fonction d'Ackermann

$$A(m, n) = \begin{cases} n + 1 & \text{si } m = 0 \\ A(m - 1, 1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{sinon} \end{cases}$$

Fonction Ackermann(  $m$  : entier,  $n$  : entier) : entier

    | Si ( $m = 0$ ) Alors  
    |     | Retourner  $n + 1$  ;

    | Fin Si  
    | Si ( $n = 0$  ET  $m > 0$ ) Alors  
    |     | Retourner Ackermann( $m - 1, 1$ ) ;

    | Simon  
    |     | Retourner Ackermann( $m - 1$ , Ackermann( $m, n - 1$ )) ;  
    | Fin Si  
Fin

## EXERCICE 2

- ❖ Soit Tab un tableau d'entiers de taille n sur lequel on veut réaliser les opérations suivantes :
  1. **Som:** qui permet de retourner la somme des éléments du tableau Tab.
  2. **Prod:** qui permet de retourner le produit des éléments du tableau Tab.
  3. **Moyenne:** qui permet de retourner la moyenne des éléments du tableau Tab.
  4. **RechElt :** qui permet de retourner l'indice de l'élément contenant une valeur donnée

# PARADIGME « DIVISER POUR RÉGNER »

- ❖ La récursivité est un outil puissant permettant de concevoir des solutions (simples) sans se préoccuper des détails algorithmiques internes
- ❖ *Paradigme Diviser pour Régner:* spécifier la solution du problème en fonction de la (ou des) solution(s) d'un (ou de plusieurs) sous-problème plus simple(s).
  - Comment trouver la solution d'un sous-problème ?
    - Ce n'est pas important car on prendra comme hypothèse que chaque appel récursif résoudra un sous-problème
    - Si on arrive à trouver une solution globale en utilisant ces hypothèses, alors ces dernières (hypothèses) sont forcément correctes, de même que la solution globale → Cela ressemble à la « démonstration par récurrence »

# PARADIGME « DIVISER POUR RÉGNER »

**Diviser**

**Régner**

**Combiner**

Problème Global

Sous Problème 1

Sous Problème 2

**Régner**

Sous Solution 1

Sous Solution 2

Sous Solution K

Décomposer le problème en k sous problème de taille moindre

Chaque sous problème sera résolu par la même décomposition récursive  
...jusqu'à l'obtention de sous problèmes triviaux

Combinaison des solutions obtenues

Solution Globale

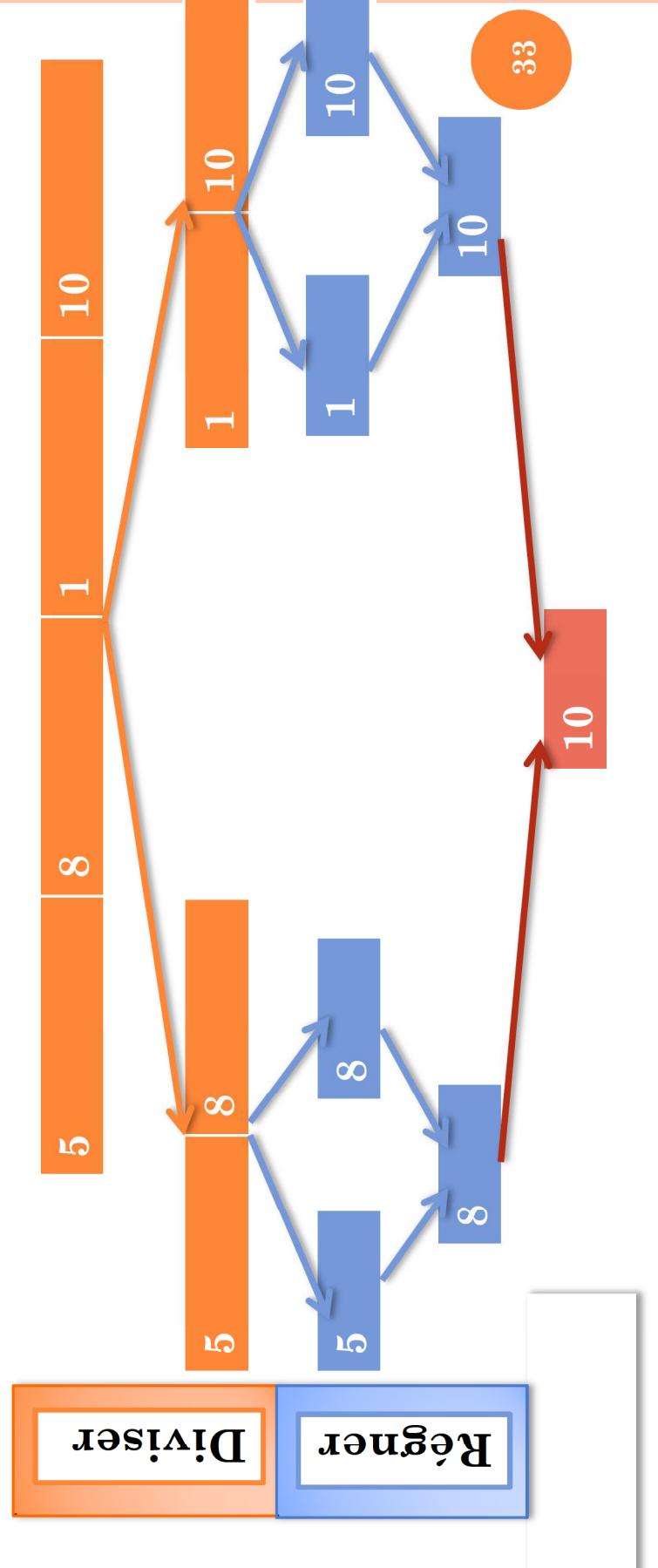
Sous Problème K

## EXERCICE 2

- ❖ Soit Tab un tableau d'entiers de taille n sur lequel on veut réaliser les opérations suivantes :
  5. **RechMax** : qui permet de retourner l'indice de l'élément le plus grand du tableau Tab
  6. **RechMin** : qui permet de retourner l'indice de l'élément le plus petit du tableau Tab
  7. **RechDicho** : qui permet de faire une recherche par dichotomie d'une valeur donnée dans le cas où le tableau Tab est trié par ordre croissant
  8. **Tri\_Fusion**: qui permet de trier le tableau Tab par la méthode de fusion.

## EXERCICE 2 : RECHERCHE MAXIMUM

- Soit Tab un tableau à n éléments, écrire une fonction récursive permettant de rechercher de l'indice du maximum dans Tab en utilisant le paradigme diviser pour régner



# EXERCICE 2 : RECHERCHE MAXIMUM

Fonction RechMax ( Tab: Tableau , indDeb, indFin:entier) : entier

Si ( indDeb = indFin) alors retourner (indDeb)

Sinon

M $\leftarrow$ (indDeb+indFin) div 2 // division du problème en 2 sous-problèmes

k1  $\leftarrow$  RechMax(Tab, indDeb, m ) // régner sur le 1er sous-problème

k2 $\leftarrow$  RechMax(Tab, m+1, indFin) // régner sur le 2ème sous-problème

// Combiner les solutions

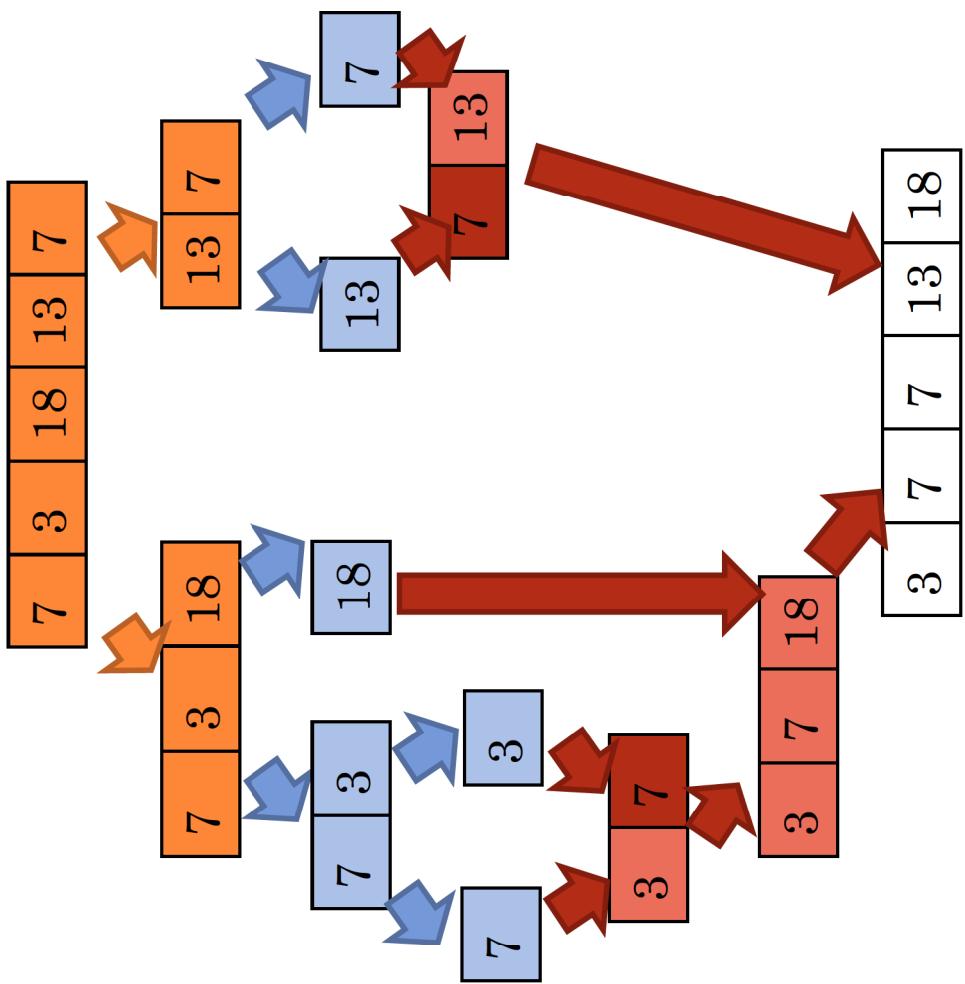
Si (Tab[k1] > Tab[k2]) alors retourner (k1)

Sinon retourner (k2)

# EXERCICE 2 : TRI PAR FUSION

## PRINCIPE

- Le principe est de trier deux tableaux de taille N/2 et ensuite de les fusionner de sorte à ce que le tableau final soit trié.



# EXERCICE 2 : TRI PAR FUSION

## PARADIGME DIVISER POUR RÉGNER

- ❖ **DIVISER:** Diviser le tableau en deux tableaux:

$$T [\text{debut}..\text{fin}] = T1[\text{debut}..\text{milieu}] + T2[\text{milieu}+1..\text{fin}]$$

- ❖ **REGNER:** trier (par fusion) les deux tableaux

- ❖ **COMBINER:** combiner les 2 tableaux de telle manière que le tableau T reste trié

## EXERCICE 2 : TRI PAR FUSION

Tri\_Fusion (T: tableau, debut, fin : entier)

Debut

Si (debut < fin) alors

milieu  $\leftarrow$  (debut + fin) /2

Tri\_Fusion(T, debut, milieu);

Tri\_fusion (T, milieu + 1, fin);

Interclasser (T, debut, milieu, fin)

FSI

Fin

# EXERCICE 2 : TRI PAR FUSION

## PROCÉDURE « INTERCLASSER »

Procédure Interclasser(T: tableau, debut, milieu, fin: entier)

Debut

Tmp: tableau temporaire du taille fin-debut+1

i  $\leftarrow$  0; i1  $\leftarrow$  debut, i2  $\leftarrow$  milieu + 1;

Tant que (i1  $\leq$  milieu) et (i2  $\leq$  fin) faire

Si (T[i1] < T[i2]) alors

    Tmp[i]  $\leftarrow$  T[i1]; i1++;

    Tmp [i]  $\leftarrow$  T[i2]; i2++;

    i++;

Tant que (i1 < milieu) faire

Tant que (i2 < fin) faire

Pour i  $\leftarrow$  debut à fin faire T[i]=tmp[i-debut]; // recopier le tableau

Fin

## EXERCICE 2 : RECHERCHE DICHOTOMIQUE

- ❖ Soit Tab un tableau trié (ordre croissant) à n éléments.
- ❖ La recherche par dichotomie compare l'élément à rechercher x avec l'élément en position m situé au milieu du sous-tableau :
  - si  $\text{Tab}[m] = x$  : on a trouvé l'élément en position m
  - si  $\text{Tab}[m] > x$  : il est impossible que x se trouve après la position m dans le tableau. Il reste à traiter uniquement la moitié inférieure du tableau
  - De même si  $\text{Tab}[m] < x$  : il reste à traiter uniquement la moitié supérieure du tableau
- ❖ On continue ainsi la recherche jusqu'à trouver l'élément ou bien aboutir sur un tableau de taille nulle, x n'est pas présent et la recherche s'arrête.

## EXERCICE 2 : RECHERCHE DICHOTOMIQUE

Fonction RechDicho(Tableau :Tableau, borneinf, bornesup, x :entier) : bool

Si (borneinf<=bornesup) alors

    mil ← (borneinf+bornesup) DIV 2 ;

    Si (Tab[mil]=x) Alors    retourner (vrai)

    Sinon

        Si (Tab[mil]>x) Alors

            Retourner (RechDicho(Tableau, borneinf, mil-1, x))

        Sinon

            Retourner(RechDicho(Tableau, mil+1, bornesup, x))

        Sinon

            Retourner (Faux)

# CONCEPTION D'UN ALGORITHME RÉCURSIF

- ❖ Dans un module récursif (procédure ou fonction) les paramètres doivent être clairement spécifiés
- ❖ Dans le corps du module, il doit y avoir:
  - un ou plusieurs cas particuliers: ce sont les cas simples qui ne nécessitent pas d'appels récursifs
  - un ou plusieurs cas généraux: ce sont les cas complexes qui sont résolus par des appels récursifs
- ❖ L'appel récursif d'un cas général doit toujours mener vers un des cas particuliers

# COMPLEXITÉ DES ALGORITHMES RÉCURSIFS

- ❖ La complexité d'un algorithme récursif se fait par la résolution d'une équation de récurrence en éliminant la récurrence par substitution de proche en proche.

► Exemple : la fonction factorielle

**Facto (n: entier): entier**

Début

Si (n=1) alors retourner 1

Sinon retourner n\*Facto (n-1);

Fin

$$T(n) = \begin{cases} c \text{ si } n = 1 \\ T(n - 1) + b \text{ sinon} \end{cases}$$

# COMPLEXITÉ DES ALGORITHMES RÉCURSIFS

- ❖ Pour calculer la solution générale de cette équation, on peut procéder par substitution :

$$\begin{aligned}T(n) &= b + T(n-1) \\&= b + [b + T(n-2)] \\&= 2b + T(n-2) \\&= 2b + [b + T(n-3)] \\&= \dots \\&= ib + T(n-i) \\&= ib + [b + T(n-i+1)] \\&= \dots \\&= (n-1)b + T(n-n+1) = nb - b + T(1) = nb - b + a \\T(n) &= nb - b + a\end{aligned}$$

43

**O (T) = O (n)**

# COMPLEXITÉ DES ALGORITHMES RÉCURSIFS

## THÉORÈME 1 DE RÉSOLUTION DE LA RÉCURRENCE

- ❖ Équations de récurrence linéaires:

$$T(n) = aT(n-1) + f(n) \quad \uparrow \quad T(n) = a^n \left( T(0) + \sum_{i=1}^n \frac{f(i)}{a^i} \right)$$

- ❖ Exemple : Factorielle

$$T(n) = \begin{cases} c & \text{si } n = 1 \\ T(n-1) + b & \text{sinon} \end{cases}$$

i.e.  $T(n) = T(n-1) + f(n)$  avec  $a = 1$ ,  $T(0) = 0$ ,  $f(n) = b$ ;

$$T(n) = \sum_{i=1}^n f(i) = \sum_{i=1}^n b = b \cdot n$$

# COMPLEXITÉ DES ALGORITHMES RÉCURSIFS

## THÉORÈME 1 DE RÉSOLUTION DE LA RÉCURRENCE

- ❖ Équations de récurrence linéaires:

$$T(n) = aT(n-1) + f(n) \quad \xrightarrow{\text{ }} \quad T(n) = a^n \left( T(0) + \sum_{i=1}^n \frac{f(i)}{a^i} \right)$$

- ❖ Exemple :  $T(n) = 2*T(n-1) + c$  avec  $T(0) = 0$

$$T(n) = 2^n \left( 0 + \sum_{i=1}^n \frac{c}{2^i} \right) = c 2^n \left( \sum_{i=1}^n \frac{1}{2^i} \right) = c 2^n (1 - 2^{-n}) = c (2^n - 1)$$

**O(T) = O(2<sup>n</sup>)**

## COMPLEXITÉ DES ALGORITHMES « DIVISER POUR RÉGNER »

- ❖ Le temps d'exécution d'un algorithme « diviser pour régner » se décompose suivant les trois étapes du paradigme de base :
  - **Diviser:** le problème en **a** sous-problèmes chacun de taille **1/b** de la taille du problème initial. Soit **D(n)** le temps nécessaire à la division du problème en sous-problèmes.
  - **Régner:** Soit **aT(n/b)** le temps de résolution des a sous-problèmes.
  - **Combiner:** Soit **C(n)** le temps nécessaire pour construire la solution finale à partir des solutions aux sous-problèmes.

$$T(n) = a \cdot T(n/b) + D(n) + C(n)$$

- ❖ Soit la fonction **f(n)** la fonction qui regroupe **D(n)** et **C(n)**. La fonction **T(n)** est alors définie :

$$T(n) = a \cdot T(n/b) + f(n)$$

# COMPLEXITÉ DES ALGORITHMES « DIVISER POUR RÉGNER »

## THÉORÈME 2 DE RÉSOLUTION DE LA RÉCURRENCE

- ❖ Pour  $f(n) = c n^k$ , on a :  $T(n) = a \cdot T(n/b) + c n^k$

$$O(T(n)) = \begin{cases} O(n \log_b a) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(f(n)) = O(n^k) & \text{si } a < b^k \end{cases}$$

# COMPLEXITÉ DES ALGORITHMES « DIVISER POUR RÉGNER »

## THÉORÈME 2 DE RÉSOLUTION DE LA RÉCURRENCE

- ❖ Pour  $T(n) = a \cdot T(n/b) + c \cdot n^k$ , on a :

$$O(T(n)) = \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(f(n)) = O(n^k) & \text{si } a < b^k \end{cases}$$

- ❖ Exercice 2: Recherche du maximum.

# EXERCICE 2 : RECHERCHE MAXIMUM

Fonction maximum ( Tab: Tableau , indDeb, indFin:entier ) : entier

Si ( indDeb = indFin ) alors retourner (indDeb)

Sinon

M $\leftarrow$ (indDeb+indFin) div 2 // division du problème en 2 sous-problèmes

k1  $\leftarrow$  maximum (Tab, indDeb, m ) // régner sur le 1er sous-problème

k2 $\leftarrow$  maximum (Tab, m+1, indFin) // régner sur le 2ème sous-problème

// Combiner les solutions

Si (Tab[k1] > Tab[k2]) alors retourner (k1)

Sinon retourner (k2)

# COMPLEXITÉ DES ALGORITHMES « DIVISER POUR RÉGNER »

## THÉORÈME 2 DE RÉSOLUTION DE LA RÉCURRENCE

- ❖ Pour  $T(n) = a \cdot T(n/b) + c \cdot n^k$ , on a :

$$O(T(n)) = \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(f(n)) = O(n^k) & \text{si } a < b^k \end{cases}$$

- ❖ Exercice 2: Recherche du maximum.

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

$$a = 2, b = 2, k = 0 \rightarrow a > b^k$$

$$\log_a a = 1$$

$$T(n) = O(n)$$

# COMPLEXITÉ DES ALGORITHMES « DIVISER POUR RÉGNER »

## THÉORÈME 2 DE RÉSOLUTION DE LA RÉCURRENCE

- ❖ Pour  $T(n) = a \cdot T(n/b) + c \cdot n^k$ , on a :

$$O(T(n)) = \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(f(n)) = O(n^k) & \text{si } a < b^k \end{cases}$$

- ❖ **Exercice 2: Recherche dichotomique**

## EXERCICE 2 : RECHERCHE DICHOTOMIQUE

Fonction RechDicho(Tab :Tableau, borneinf, bornesup, x :entier) : bool

Si (borneinf<=bornesup) alors

    mil  $\leftarrow$  (borneinf+bornesup) DIV 2 ;

    Si (Tab[mil]=x) Alors    retourner (vrai)

    Sinon

        Si (Tab[mil]>x) Alors

            Retourner (RechDicho(Tab, borneinf, mil-1, x))

        Sinon

            Retourner(RechDicho(Tab, mil+1, bornesup, x))

        Sinon

            Retourner (Faux)

# COMPLEXITÉ DES ALGORITHMES « DIVISER POUR RÉGNER »

## THÉORÈME 2 DE RÉSOLUTION DE LA RÉCURRENCE

- ❖ Pour  $T(n) = a \cdot T(n/b) + c \cdot n^k$ , on a :

$$O(T(n)) = \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(f(n)) = O(n^k) & \text{si } a < b^k \end{cases}$$

- ❖ Exercice 2: Recherche dichotomique

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

$$a = 1, b = 2, k = 0 \rightarrow a = b^k$$

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

# COMPLEXITÉ DES ALGORITHMES « DIVISER POUR RÉGNER »

## THÉORÈME 2 DE RÉSOLUTION DE LA RÉCURRENCE

- ❖ Pour  $T(n) = a.T(n/b) + c.n^k$ , on a :

$$O(T(n)) = \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(f(n)) = O(n^k) & \text{si } a < b^k \end{cases}$$

- ❖ Exercice 2: Tri par Fusion

## EXERCICE 2 : TRI PAR FUSION

Tri\_Fusion (T: tableau, debut, fin : entier)

Debut

Si (debut < fin) alors

milieu  $\leftarrow$  (debut + fin) /2

Tri\_Fusion(T, debut, milieu);

Tri\_fusion (T, milieu + 1, fin);

Interclasser (T, debut, milieu, fin)

FSI

Fin

# EXERCICE 2 : TRI PAR FUSION

## PROCÉDURE « INTERCLASSER »

Procédure Interclasser(T: tableau, debut, milieu, fin: entier)

Debut

Tmp: tableau temporaire du taille fin-debut+1

i  $\leftarrow$  0; i1  $\leftarrow$  debut, i2  $\leftarrow$  milieu + 1;

Tant que (i1  $\leq$  milieu) et (i2  $\leq$  fin) faire

Si (T[i1] < T[i2]) alors

    Tmp[i]  $\leftarrow$  T[i1]; i1++;

    Tmp [i]  $\leftarrow$  T[i2]; i2++;

    i++;

Tant que (i1 < milieu) faire

Tant que (i2 < fin) faire

Pour i  $\leftarrow$  debut à fin faire T[i]=tmp[i-debut]; // recopier le tableau

Fin

# COMPLEXITÉ DES ALGORITHMES « DIVISER POUR RÉGNER »

## THÉORÈME 2 DE RÉSOLUTION DE LA RÉCURRENCE

- ❖ Pour  $T(n) = a \cdot T(n/b) + c \cdot n^k$ , on a :

$$O(T(n)) = \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(f(n)) = O(n^k) & \text{si } a < b^k \end{cases}$$

- ❖ Exercice 2: Tri par Fusion

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

$$a = 2, b = 2, k = 1 \rightarrow a = b^k$$

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

# COMPLEXITÉ DES ALGORITHMES « DIVISER POUR RÉGNER »

## THÉORÈME 2 DE RÉSOLUTION DE LA RÉCURRENCE

- ❖ Pour  $T(n) = a \cdot T(n/b) + c \cdot n^k$ , on a :

$$O(T(n)) = \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(f(n)) = O(n^k) & \text{si } a < b^k \end{cases}$$

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

- $a = 9$ ,  $b = 3$ ,  $k = 1$   
→  $a > b^k$   
→  $\log_b a = 2$   
→  $T(n) = O(n^2)$

# COMPLEXITÉ DES ALGORITHMES « DIVISER POUR RÉGNER »

## THÉORÈME 2 DE RÉSOLUTION DE LA RÉCURRENCE

- ❖ Pour  $T(n) = a \cdot T(n/b) + c \cdot n^k$ , on a :

$$O(T(n)) = \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(f(n)) = O(n^k) & \text{si } a < b^k \end{cases}$$

$$T(n) = 3T\left(\frac{n}{4}\right) + n \log n$$

■  $a = 3$ ,  $b = 4$ ,  $k = ?$ ?  
or  $n < n \log n < n^2 \rightarrow 1 < k < 2$   
 $\rightarrow 4 < b^k < 16 \rightarrow a < b^k$   
 $\rightarrow T(n) = O(f(n)) = O(n \log n)$

## THÉORÈME 3 DE RÉSOLUTIONS DE RÉCURRENCE

- ❖ Équations de récurrence linéaires sans second membre ( $f(n) = \text{cte}$ )

$$T(n) = a_1 T(n-1) + a_2 T(n-2) + a_3 T(n-3) + \dots + a_k T(n-k) + cst$$

- ❖ A une telle équation, on peut associer un polynôme:

$$P(x) = x^k - a_1 x^{k-1} - a_2 x^{k-2} - \dots - a_k$$

- La résolution de ce polynôme nous donne m racines ri ( avec  $m \leq k$ ).

- ❖ La solution de l'équation de récurrence est ainsi donnée par :

$$T(n) = c_1 r_1^n + c_2 r_2^n + c_3 r_3^n + \dots + c_m r_m^n$$

- Cette solution est en général exponentielle

# COMPLEXITÉ DES ALGORITHMES « DIVISER POUR RÉGNER »

## THÉORÈME 3 DE RÉSOLUTIONS DE RÉCURRENCE

### ❖ Exercice 1 : La suite de Fibonacci

$$F(n) = \begin{cases} 1, & n=0 \text{ ou } n=1, \\ F(n-1) + F(n-2), & n \geq 2. \end{cases}$$

□  $T(n) = T(n-1) + T(n-2)$

□ On pose  $P(x) = X^2 - X - 1$

$$\begin{aligned} \rightarrow r_1 &= \frac{1+\sqrt{5}}{2} \text{ et } r_2 = \frac{1-\sqrt{5}}{2} \\ T(n) &= a r_1^n + b r_2^n = a \left( \frac{1+\sqrt{5}}{2} \right)^n + b \left( \frac{1-\sqrt{5}}{2} \right)^n \\ \rightarrow T(n) &= O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right) \end{aligned}$$

# SOURCES DE CE COURS

- Frédéric Vivien, Algorithmique avancée, École Normale Supérieure de Lyon, 2002., pp. 93. Disponible sur <http://perso.ens-lyon.fr/frederic.vivien/Enseignement/Algorithmique/2001-2002/Cours.pdf>
- Slim Msfar, Algorithmique et Complexité, 2012, pp 104. Disponible sur <http://p835.phponet.org/testremorque/upload/catalogue/coursalgorithmi.pdf>
- Walid-Khaled Hidouci, La récursivité, École nationale Supérieure d'Informatique, pp 107. Disponible sur [hidouci.esi.dz/algo/cours\\_recurssive.pdf](http://hidouci.esi.dz/algo/cours_recurssive.pdf)
- La récursivité, Cours d' Algorithmique et Structures de données dynamiques, École nationale Supérieure d'Informatique, Année Universitaire 2009-2010.