COURS DE NSI (Numérique et Sciences Informatiques)

Classe : NSIT^le

séquence 5

Mr BANANKO K.

LYCÉE INTERNATIONAL COURS LUMIÈRE

Méthode diviser pour régner

Manuel numérique/ hachette Education
Page 122
'les algorithmes du type « diviser pour régner »'

Objectifs :

Contenus	Capacités attendues	Commentaires
Méthode « diviser pour régner »	Écrire un algorithme utilisant la méthode « diviser pour régner ».	La rotation d'une image bitmap d'un quart de tour avec un coût en mémoire constant est un bon exemple. L'exemple du tri fusion permet également d'exploiter la récursivité et d'exhiber un algorithme de coût en $n \log_2 n$ dans les pires des cas.

Nous avons vu en première deux algorithmes de tris assez naturels, mais peu efficaces: le tri par insertion et le tri par sélection. Cette année, nous allons étudier un algorithme beaucoup plus efficace et très utilisé inventé par John Von Neumann en 1945: le tri par fusion. Cet algorithme nous permettra d'illustrer la méthode diviser pour régner que nous avions déjà vue lors de la recherche dichotomique.

I- Diviser pour régner

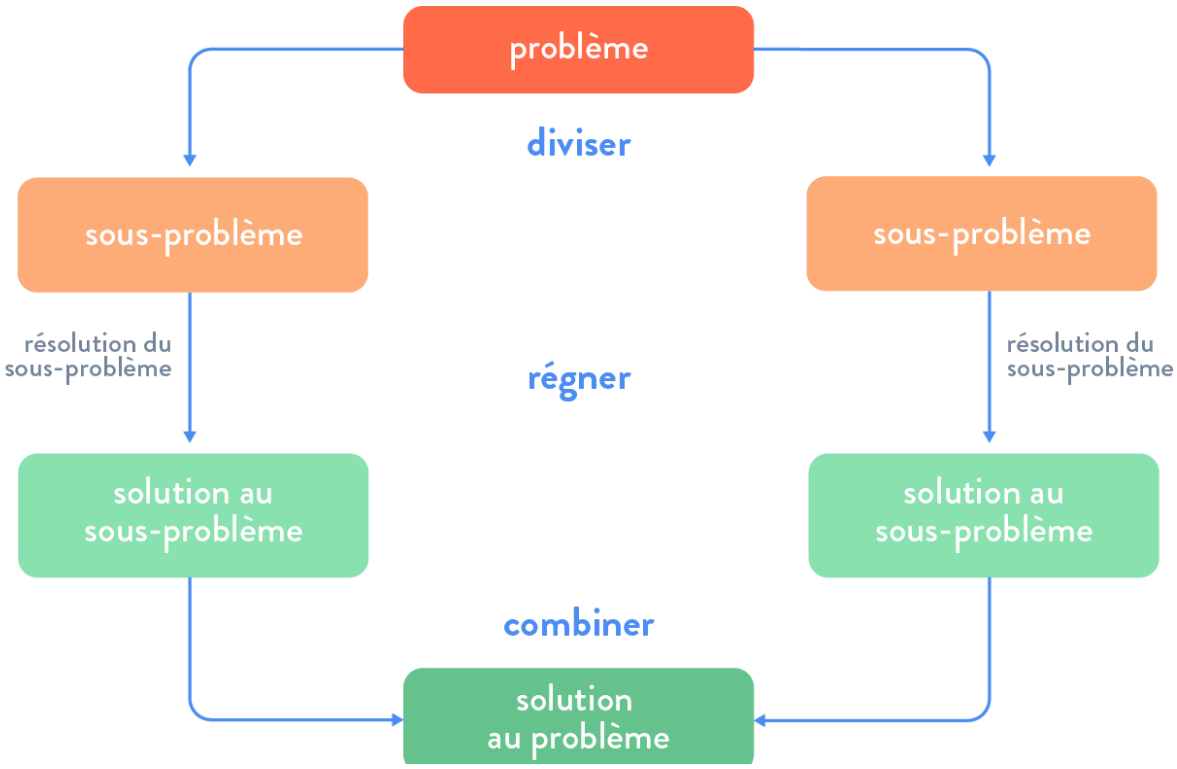
Le diviser pour régner est une méthode algorithmique basée sur le principe suivant :

On prend un problème (généralement complexe à résoudre), on divise ce problème en une multitude de petits problèmes, l'idée étant que les "petits problèmes" seront plus simples à résoudre que le problème original. Une fois les petits problèmes résolus, on recombine les "petits problèmes résolus" afin d'obtenir la solution du problème de départ.

Le paradigme "diviser pour régner" repose donc sur 3 étapes :

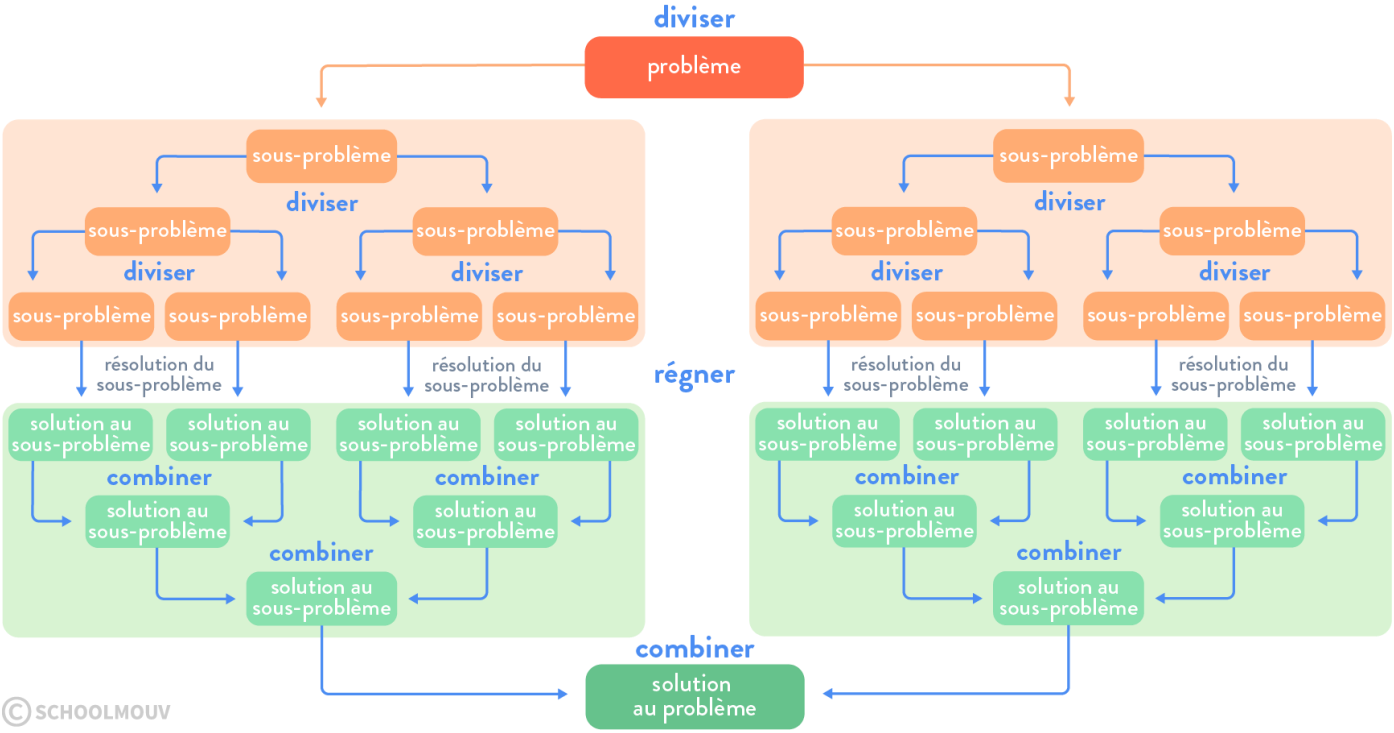
- **DIVISER** : le problème d'origine est divisé en un certain nombre de sous-problèmes
- **RÉGNER** : on résout les sous-problèmes (les sous-problèmes sont plus faciles à résoudre que le problème d'origine)

- **COMBINER** : les solutions des sous-problèmes sont combinées afin d'obtenir la solution du problème d'origine.



Ces sous-problèmes peuvent eux-mêmes être décomposés en sous-problèmes selon une logique récursive. En cas de divisions multiples, il y aura, après les résolutions des sous-problèmes individuels, autant d'étapes de recombinaison qu'il a eu d'étapes de division.

Le schéma ci-après illustre le processus avec plusieurs niveaux de divisions et de recombinaisons.



Les algorithmes basés sur le paradigme "**diviser pour régner**" sont très souvent des algorithmes récursifs.

Nous allons maintenant étudier un de ces algorithmes basés sur le principe diviser pour régner:
le tri-fusion

II- Tri-fusion

1- Rappel du tri par insertion et le tri par sélection

♣ Algorithme du tri par insertion

Entrons tout de suite dans le vif du sujet, voici l'algorithme du tri par insertion :

```
VARIABLE
t : tableau d'entiers
i : nombre entier
j : nombre entier
k : nombre entier
DEBUT
j←2
tant que j<=longueur(t):   //boucle 1
  i←j-1
  k←t[j]
  tant que i>0 et que t[i]>k:   //boucle 2
    t[i+1]←t[i]
    i←i-1
  fin tant que
  t[i+1]←k
  j←j+1
fin tant que
FIN
```

Activité

Poursuivez le travail commencé ci-dessous (attention de bien donner l'état du tableau à chaque étape)

t = [27, 10, 12, 8, 11]

j=2
j<longueur(t) => on "entre" dans la boucle 1
i←j-1 (i=1)
k←t[j] (k=10)
i>0 et t[1]>k => on "entre" dans la boucle 2
t[i+1]←t[i] (t[2]=27)=> t=[27, 27, 12, 8, 11]
i←i-1(i=0)
retour au début de la boucle 2

↓

j=2
i=0
k=10
i=0 => on n'entre pas dans la boucle 2
t[i+1]←k (t[1]=10) => t=[10, 27, 12, 8, 11]
j←j+1 (j=3)
retour au début de la boucle 1

↓

j=3
j<longueur(t) => on "entre" dans la boucle 1
i←j-1 (i=2)
k←t[j] (k=12)
i>0 et t[2]>k => on "entre" dans la boucle 2
t[i+1]←t[i] (t[3]=27)=> t=[10, 27, 27, 8, 11]
i←i-1 (i=1)
retour au début de la boucle 2

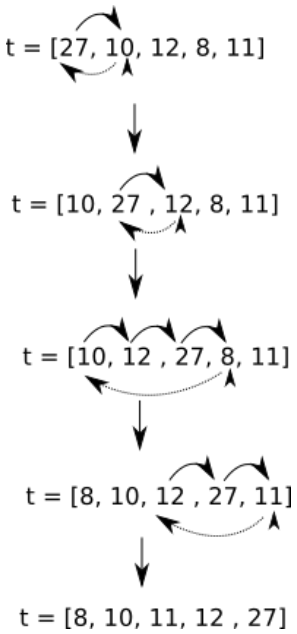
↓

j=3
i=1
k=12
t[1]<k => on n'entre pas dans la boucle 2
t[i+1]←k (t[2]=12)=> t=[10, 12, 27, 8, 11]
j←j+1 (j=4)
retour au début de la boucle 1

↓

.....

On peut résumer le principe de fonctionnement de l'algorithme de tri par insertion avec le schéma suivant :



L'algorithme de tri par insertion a une complexité en $O(n^2)$. On parle aussi de complexité quadratique.

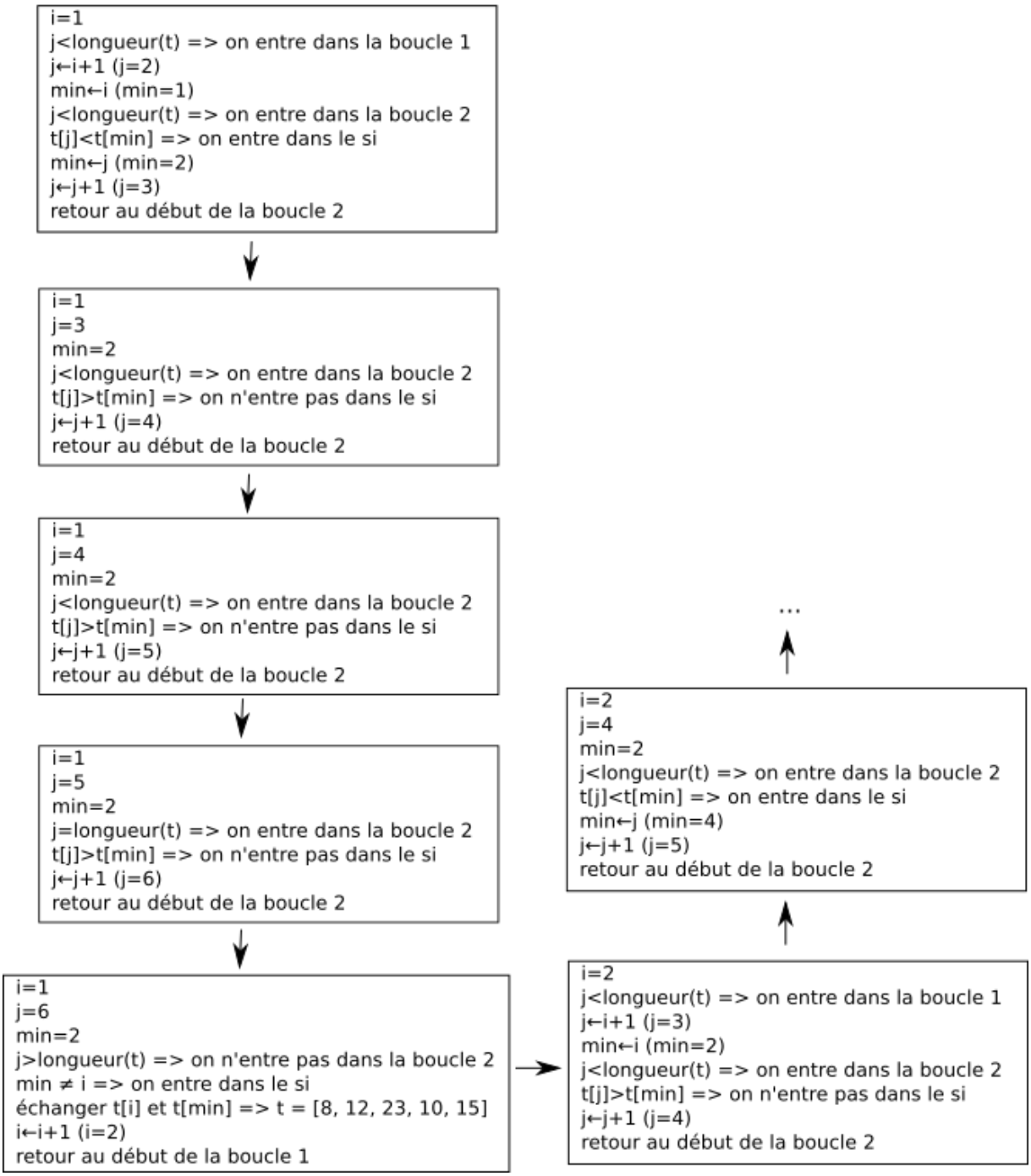
♣ Algorithme du tri par sélection

```
VARIABLE
t : tableau d'entiers
i : nombre entier
min : nombre entier
j : nombre entier
DEBUT
i←1
tant que i<longueur(t):   //boucle 1
  j←i+1
  min←i
  tant que j<=longueur(t):   //boucle 2
    si t[j]<t[min]:
      min←j
    fin si
    j←j+1
  fin tant que
  si min≠i :
    échanger t[i] et t[min]
  fin si
  i←i+1
fin tant que
FIN
```

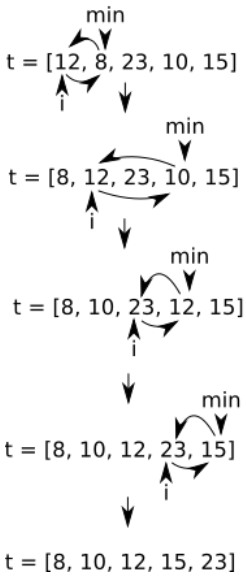
Activité 3

Poursuivez le travail commencé ci-dessous (attention de bien donner l'état du tableau)

t = [12, 8, 23, 10, 15]



On peut résumer le principe de fonctionnement de l'algorithme de tri par sélection avec le schéma suivant :



Les algorithmes de tri par sélection et de tri par insertion ont tous les deux une complexité quadratique ($O(n^2)$) , dans le meilleur des cas le meilleur des cas et en moyenne.

Il est important de bien avoir conscience de l'impact de ces complexités sur l'utilisation des algorithmes : si vous doublez la taille du tableau, vous doublerez le temps d'exécution d'un algorithme de complexité linéaire, en revanche vous quadruplerez le temps d'exécution d'un algorithme de complexité quadratique.

2- Présentation du tri-fusion

Nous avons déjà étudié des algorithmes de tri : **le tri par insertion et le tri par sélection**. Nous allons maintenant étudier une nouvelle méthode de tri, le tri-fusion. Comme pour les algorithmes déjà étudiés, cet algorithme de tri fusion prend en entrée un tableau non trié et donne en sortie, le même tableau, mais trié.

a- Première version de l'algorithme

Activité 1

Voici l'algorithme de tri fusion

Étudiez cet algorithme :

```
VARIABLE
A : tableau d'entiers
L : tableau d'entiers
R : tableau d'entiers
p : entier
q : entier
r : entier
n1 : entier
n2 : entier


DEBUT

FUSION (A, p, q, r):
  n1 ← q - p + 1
  n2 ← r - q
  créer tableau L[1..n1+1] et R[1..n2+1]
  pour i ← 1 à n1:
    L[i] ← A[p+i-1]
  fin pour
  pour j ← 1 à n2:
    R[j] ← A[q+j]
  fin pour
  L[n1+1] ← ∞
  R[n2+1] ← ∞
  i ← 1
  j ← 1
  pour k ← p à r:
    si L[i] ⩽ R[j]:
      A[k] ← L[i]
      i ← i + 1
    sinon:
      A[k] ← R[j]
      j ← j + 1
    fin si
  fin pour
fin FUSION

TRI-FUSION(A, p, r):
  si p < r:
    q = (p + r) / 2
    TRI-FUSION(A, p, q)
    TRI-FUSION(A, q+1, r)
    FUSION(A, p, q, r)
  fin si
fin TRI-FUSION
FIN
```

Pour trier un tableau A, on fait l'appel initial ***TRI-FUSION(A, 1, A.longueur)***

RAPPEL : ATTENTION, EN ALGORITHMIQUE, LES INDICES DES TABLEAUX COMMENCENT A 1

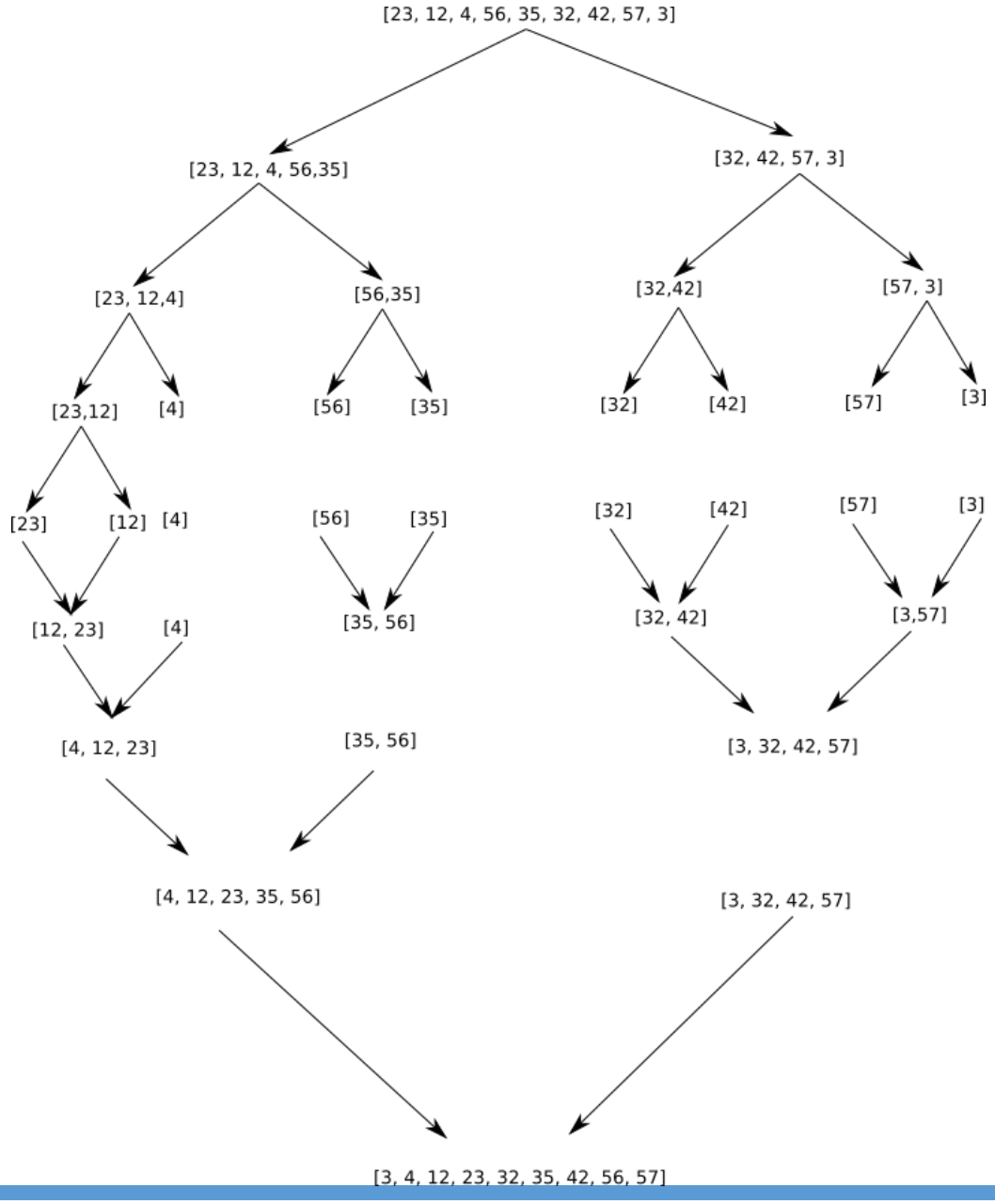
Cet algorithme est un peu difficile à appréhender, on notera qu'il est composé de deux fonctions FUSION et TRI-FUSION (fonction récursive).

La fonction TRI-FUSION assure la phase "**DIVISER**" et la fonction FUSION assure les phases "**RÉGNER**" et "**COMBINER**".

Voici un exemple d'application de cet algorithme sur le tableau

A = [23, 12, 4, 56, 35, 32, 42, 57, 3] :

Activité 2 : Étudiez attentivement le schéma ci-dessous afin de mieux comprendre le principe du tri-fusion (identifiez bien les phases "**DIVISER**" et "**COMBINER**").



On remarque que dans le cas du tri-fusion, la phase "**RÉGNER**" se réduit à sa plus simple expression, en effet, à la fin de la phase "**DIVISER**", nous avons à trier des tableaux qui comportent un seul élément, ce qui est évidemment trivial.

La fusion des 2 tableaux déjà triés est simple, prenons comme exemple la dernière fusion entre le tableau [4, 12, 23, 35, 56] et le tableau [3, 32, 42, 57] (le principe est identique pour toutes les fusions) :

Soit T le tableau issu de la fusion du tableau B = [4, 12, 23, 35, 56] et du tableau C = [3, 32, 42, 57] (on donne des noms aux tableaux uniquement pour essayer de rendre l'explication la plus claire possible).

- On considère le premier élément du tableau B (4) et le premier élément du tableau C (3) : 3 est inférieur à 4, on place 3 dans le tableau T et on le supprime du tableau C. Nous avons donc alors T = [3], B = [4, 12, 23, 35, 56] et C = [32, 42, 57].
- On recommence ensuite à comparer le premier élément du tableau B (4) et le premier élément du tableau C (32) : 4 est inférieur à 32, on place 4 dans le tableau T et on le supprime du tableau B. Nous avons donc alors T = [3, 4], B = [12, 23, 35, 56] et C = [32, 42, 57].
- On compare le premier élément du tableau B (12) et le premier élément du tableau C (32) : 12 est inférieur à 32, on place 12 dans le tableau T et on le supprime du tableau B. Nous avons donc alors T = [3, 4, 12], B = [23, 35, 56] et C = [32, 42, 57].
- On compare le premier élément du tableau B (23) et le premier élément du tableau C (32) : 23 est inférieur à 32, on place 23 dans le tableau T et on le supprime du tableau B. Nous avons donc alors T = [3, 4, 12, 23], B = [35, 56] et C = [32, 42, 57].
- On compare le premier élément du tableau B (35) et le premier élément du tableau C (32) : 32 est inférieur à 35, on place 32 dans le tableau T et on le supprime du tableau A. Nous avons donc alors T = [3, 4, 12, 23, 32], B = [35, 56] et C = [42, 57].
- On compare le premier élément du tableau B (35) et le premier élément du tableau C (42) : 35 est inférieur à 42, on place 35 dans le tableau T et on le supprime du tableau A. Nous avons donc alors T = [3, 4, 12, 23, 32, 35], B = [56] et C = [42, 57].
- On compare le premier élément du tableau B (56) et le premier élément du tableau C (42) : 42 est inférieur à 56, on place 42 dans le tableau T et on le supprime du tableau A. Nous avons donc alors T = [3, 4, 12, 23, 32, 35, 42], B = [56] et C = [57].
- On compare le premier élément du tableau B (56) et le premier élément du tableau C (57) : 56 est inférieur à 57, on place 56 dans le tableau T et on le supprime du tableau B. Nous avons donc alors T = [3, 4, 12, 23, 32, 35, 42, 56], B = [] et C = [57].
- Le tableau B est vide, il nous reste juste à placer le seul élément qui reste dans C (57) dans T : T = [3, 4, 12, 23, 32, 35, 42, 56, 57], B = [] et C = []. La fusion est terminée.

Activité 3

Reprenez tout le raisonnement qui vient d'être fait sur le tableau T = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]. Vous n'hésiterez pas à faire un schéma et à expliquer la fusion de 2 tableaux triés.

b- Deuxième version de l'algorithme

Algorithme de fusion

Voici l'algorithme de fusion de deux tableaux triés en un seul.

Tout d'abord en pseudo-code:

```
fonction fusion(tbl1: Tableau, tbl2: Tableau)
    // tbl1 et tbl2 sont deux tableaux triés
// Initialisation
i1 &lt;- 0   // indice du 1er tableau
i2 &lt;- 0   // indice du 2e tableau
tbl &lt;- []   // liste vide destinée à accueillir les éléments triés

// Boucle

TANT QUE l'on a pas atteint la fin d'un des tableaux

SI tbl1[i1] <= tbl2[i2] ALORS

Insérer tbl1[i1] à la fin de tbl

incrémenter i1

SINON

Insérer tbl2[i2] à la fin de tbl

incrémenter i2

FIN SI

FIN TANT QUE

// Finalisation
// Insérer les éléments restants du tableau non vide à la fin de tbl
SI i1 &lt; longueur de tbl1 ALORS
    Insérer tous les éléments restants de tbl1 à la fin de tbl
SINON SI i2 &lt; longueur de tbl2 ALORS
    Insérer tous les éléments restants de tbl2 à la fin de tbl

RENVOYER tbl
```

Et voici une implémentation en python:

```python
def fusion (tbl1: list, tbl2: list) -> list:
    # Initialisation
    N1, N2 = len(tbl1), len(tbl2)
    i1 = 0
    i2 = 0
    tbl = []

    # Boucle sur les deux tableaux
    while (i1 < N1) and (i2 < N2):
        x1, x2 = tbl1[i1], tbl2[i2]
        # si x1 < x2  on ajoute l'élément x1 à tbl
        if x1 <= x2:
            tbl.append(x1)
            i1 = i1 + 1
        # sinon on ajoute l'élément x2
        else:
            tbl.append(x2)
            i2 = i2 + 1

    # Finalisation: On ajoute les éléments restants du tableau non vide restant
    # Si tbl1 n'a pas été entièrement vidé, on ajoute ses éléments restants
    if i1 < N1:
        for i in range(i1, N1):
            tbl.append(tbl1[i])
    # Sinon on ajoute les éléments de tbl2 restants
    elif i2 < N2:
        for i in range(i2, N2):
            tbl.append(tbl2[i])

    return tbl
```

Un petit test dans la console ipython permet de vérifier sur un cas simple la fusion:

```
>>> fusion([3,6,8], [2,5,7,12])
[2, 3, 5, 6, 7, 8, 12]
```

Algorithme de tri fusion

Voici l'algorithme récursif de tri fusion qui utilise la fonction fusion définie précédemment.

Tout d'abord en pseudo-code, on retrouve des techniques de découpage du tableau en deux avec des divisions entières // vues dans la recherche dichotomique.

```
fonction tri_fusion(tbl: Tableau)
    N <- Longueur de tbl

    // Cas terminal: une liste de un élément est forcément triée
    SI N == 1 ALORS
        RENVOYER tbl
    FIN SI

    // Recursion sur les deux demi-tableaux sinon
    tbl1 <- liste des N//2 premiers éléments de tbl
    tbl2 <- liste des N//2 derniers éléments de tbl

    // Renvoi des la fusion des deux tableaux
    RENVOYER fusion(tri_fusion(tbl1), tri_fusion(tbl2))
```

Et voici une implémentation en python qui utilise les listes en compréhension:

```python
def tri_fusion (tbl: list) -> list:
    N = len(tbl)
    # cas de base: un tableau de zéro ou un élément est forcément trié!
    if N < 2:
        return tbl

    # on coupe le tableau en deux
    tbl1 = [tbl[i] for i in range(N//2)]
    tbl2 = [tbl[i] for i in range(N//2, N)]

    # appels récursifs
    return fusion(tri_fusion(tbl1), tri_fusion(tbl2))
```

On fait un petit test sur une liste quelconque.

```
>>> tri_fusion([0, 25, 36, 41, 1, 465, 2, 3, 987])
[0, 1, 2, 3, 25, 36, 41, 465, 987]
```

3- Complexité

Nous avons vu que le tri par insertion et tri par sélection ont tous les deux une complexité $O(n^2)$. Qu'en est-il pour le tri-fusion ?

Le calcul rigoureux de la complexité de cet algorithme sort du cadre de ce cours. Mais, en remarquant que la première phase (DIVISER) consiste à "**couper**" les tableaux en deux plusieurs fois de suite, intuitivement, on peut dire qu'un logarithme base 2 doit intervenir. La deuxième phase consiste à faire des comparaisons entre les premiers éléments de chaque tableau à fusionner, on peut donc supposer que pour un tableau de n éléments, on aura n comparaisons. En combinant ces 2 constations on peut donc dire que la complexité du tri-fusion est en ***O(n.log(n))*** (encore une fois la "**démonstration**" proposée ici n'a rien de rigoureux).
La comparaison des courbes de la fonction n^2 (en rouge) et ***n.log(n)*** (en bleu) :

nous montre que l'algorithme de tri-fusion est plus "efficace" que l'algorithme de tri par insertion ou que l'algorithme de tri par sélection.

Exercice du BAC

- Sujet 2 2021 Exercice 4
- Sujet 4 2021 Exercice 5
- Sujet 2 2022 Exercice 4
- Sujet 3 2022 Exercice 5