ALGO 4/5

studi

Auteur : Sébastien Inion

https://github.com/SebInfo/AlgoStudi4

LE PARADIGME DIVISER POUR RÉGNER

- Un paradigme est une approche ou un modèle de pensée qui définit un ensemble de concepts, de méthodes, de pratiques et de principes pour résoudre des problèmes dans un domaine spécifique
- C'est donc une autre manière d'aborder un problème.
- Diviser pour régner fait appel à la récursivité.
- Il y a trois étapes :
  - **Diviser** : le problème est décomposé en sous problèmes (instances plus petites du même problème)
  - **Régner** : On règne sur les sous problèmes en les résolvant de manière récursive.
  - **Combiner** : On rassemble les solutions des sous-problèmes pour produire la solution finale.

LE TRI FUSION

- Nous allons adopter ce paradigme pour trier notre tableau d'entiers.
- Cela donne un autre algo de tri : le tri par fusion
    - Diviser : On va diviser le tableau à trier en deux tableaux à trier
    - Régner : On va trier les deux tableaux de manière récursive. On va donc repartager en deux.
    - Combiner : on va fusionner pour produire le tableau trié

LE TRI FUSION

- On a donc deux algos à écrire
  - Un algo pour le tri en lui même et les appels récursifs
  - Un algo pour la fusion
- Algo pour le tri:

```
fonction triFusion(tableau)
    si la longueur du tableau <= 1
        retourner tableau

    moitie = longueur du tableau / 2
    moitieGauche = triFusion(première moitié du tableau)
    moitieDroite = triFusion(deuxième moitié du tableau)

    tableauTrié = fusionner(moitieGauche, moitieDroite)
    retourner tableauTrié
```

LE TRI FUSION

- On doit fusionner **deux tableaux triés.**
  Imaginons pour simplifier deux tas de cartes **triés** : la carte la plus faible est en haut.
- On doit donc pour trier choisir la carte la plus faible parmi les deux tas et la placer face cacher sur la table. (on forme donc un troisième tas)
- On répète cela jusqu'à épuisement d'un des deux tas
- On place alors les cartes du tas qui reste au dessus du tas final.

LE TRI FUSION

- Algo pour la fusion :

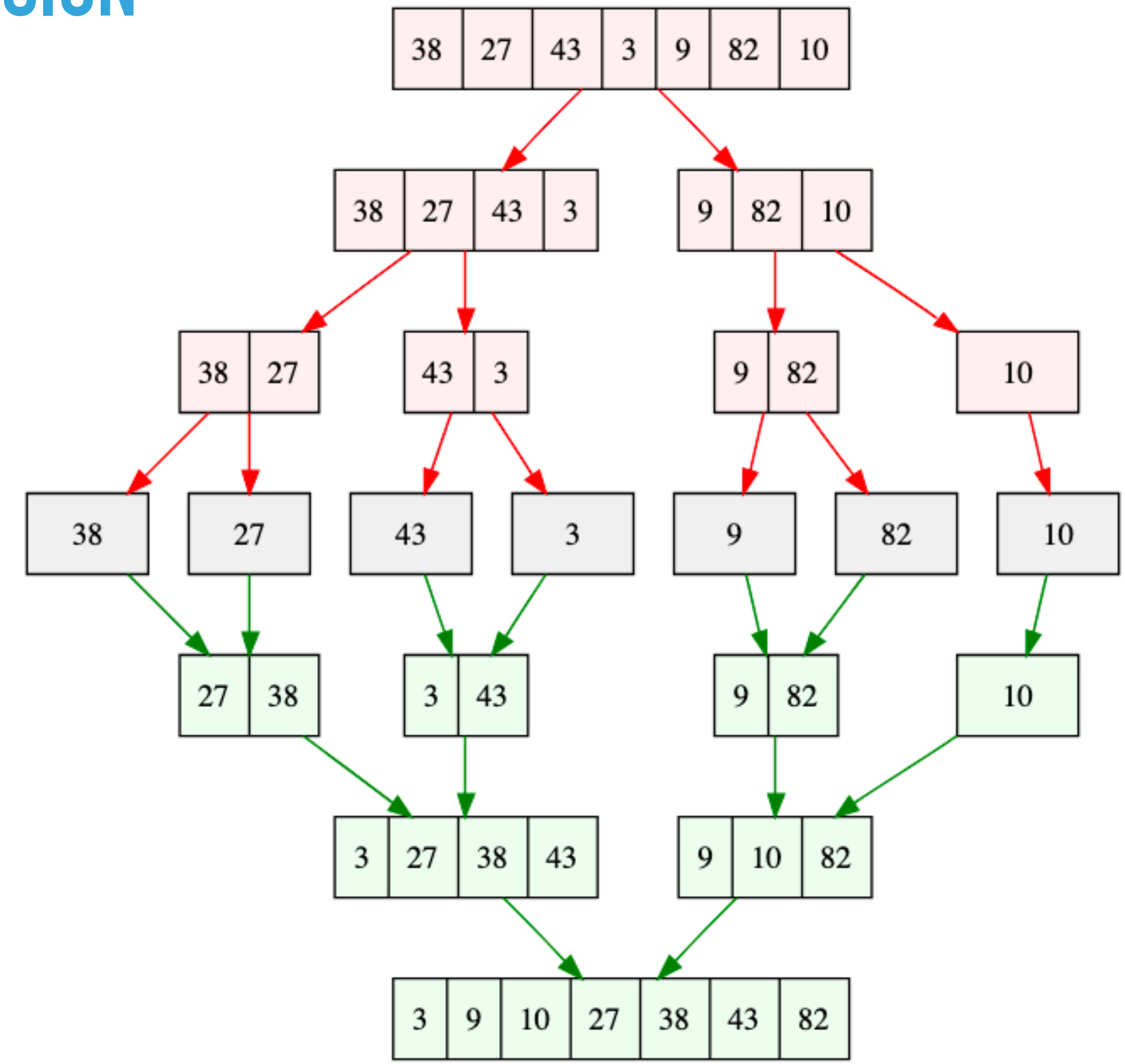
```
fonction fusionner(moitieGauche, moitieDroite)
    tableauTrié = tableau vide
    pointeurGauche = 0
    pointeurDroite = 0

    tant que pointeurGauche < longueur de moitieGauche et pointeurDroite < l
        si moitieGauche[pointeurGauche] <= moitieDroite[pointeurDroite]
            ajouter moitieGauche[pointeurGauche] à tableauTrié
            incrémenter pointeurGauche
        sinon
            ajouter moitieDroite[pointeurDroite] à tableauTrié
            incrémenter pointeurDroite

    ajouter les éléments restants de moitieGauche à tableauTrié
    ajouter les éléments restants de moitieDroite à tableauTrié

    retourner tableauTrié
```

LE TRI FUSION



IMPLÉMENTATION EN PYTHON

```
fonction triFusion(tableau)
    si la longueur du tableau <= 1
        retourner tableau

    moitie = longueur du tableau / 2
    moitieGauche = triFusion(première moitié du tableau)
    moitieDroite = triFusion(deuxième moitié du tableau)

    tableauTrié = fusionner(moitieGauche, moitieDroite)
    retourner tableauTrié
```

```python
def tri_fusion (tbl: list) -> list:
    n = len(tbl) # taille du tableau
    if len(tbl) <= 1: # cas de base (un tableau vide ou avec un élement est trié)
        return tbl

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

    # On règne avec les appels récursifs et on combine
    return fusion(tri_fusion(tbl1), tri_fusion(tbl2))
```

IMPLÉMENTATION EN PYTHON

```
fonction fusionner(moitieGauche, moitieDroite)
    tableauTrié = tableau vide
    pointeurGauche = 0
    pointeurDroite = 0

    tant que pointeurGauche < longueur de moitieGauche et pointeurDroite < l
        si moitieGauche[pointeurGauche] <= moitieDroite[pointeurDroite]
            ajouter moitieGauche[pointeurGauche] à tableauTrié
            incrémenter pointeurGauche
        sinon
            ajouter moitieDroite[pointeurDroite] à tableauTrié
            incrémenter pointeurDroite

    ajouter les éléments restants de moitieGauche à tableauTrié
    ajouter les éléments restants de moitieDroite à tableauTrié

    retourner tableauTrié
```

```python
def fusion (tbl1: list, tbl2: list) -> list:
    n1, n2 = len(tbl1), len(tbl2)
    i = 0
    j = 0
    tbl = []

    # Boucle sur les deux tableaux
    while (i < n1) and (j < n2):
        x1, x2 = tbl1[i], tbl2[j]
        # si x1 < x2  on ajoute l'élément x1 à tbl
        if x1 <= x2:
            tbl.append(x1)
            i = i + 1
        # sinon on ajoute l'élément x2
        else:
            tbl.append(x2)
            j = j + 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 i < n1:
        for i in range(i, n1):
            tbl.append(tbl1[i])
    # Sinon on ajoute les éléments de tbl2 restants
    elif j < n2:
        for i in range(j, n2):
            tbl.append(tbl2[i])

    return tbl
```

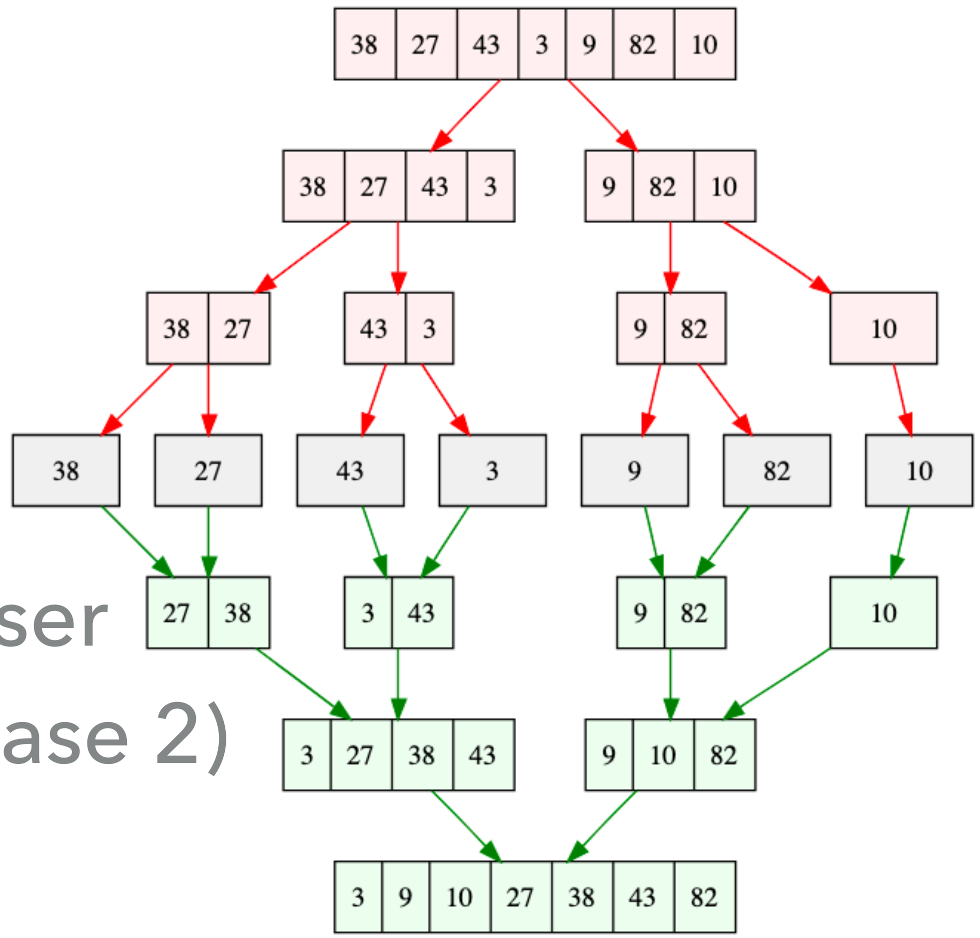
LA COMPLEXITÉ DE L'ALGO

LE TRI FUSION

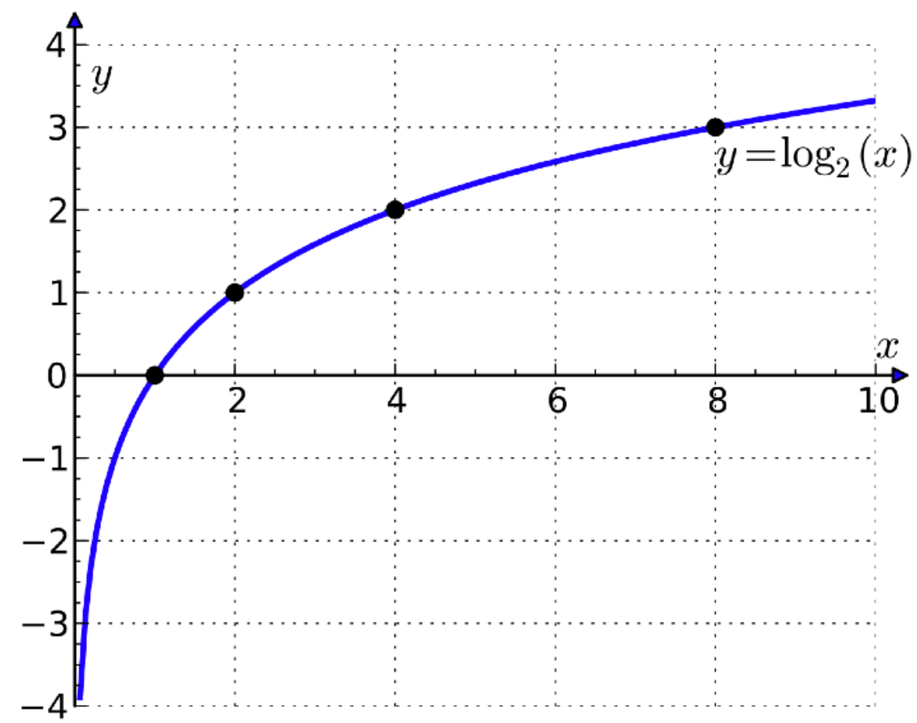
- **Le tri par sélection** à une complexité quadratique O(n2) dans le pire des cas, le meilleur des cas et en moyenne
- **Le tri par insertion** à une complexité linéaire O(n) dans le meilleur des cas, et quadratique O(n2)dans le pire des cas et en moyenne
- **Le tri à bulles** à une complexité linéaire O(n) dans le meilleur des cas, et quadratique O(n2) dans le pire des cas et en moyenne
- Quelle est la complexité du **tri fusion** ?

LE TRI FUSION

- **La fusion** à une complexité de O(n) en effet la fusion doit comparer au pire n élément(s) comme les deux tableaux sont déjà triés.
- La fusion c'est les flèches vertes
- Mais elle doit le faire autant de fois qu'on a crée de sous tableaux
- Donc autant de fois qu'on peut diviser par 2 - > log(n) (ici c'est un log en base 2)



LE TRI FUSION



- **Log(n) est très intéressant :**
  - Log(8) = 3
  - Log(99) = 6
  - Log(50 000) = 15
- **On a donc une complexité en $O(n\log(n))$** dans le pire des cas, dans le meilleur des cas et en moyenne.
- C’est donc un algorithme plus performant dans les cas moyens et les pires des cas.

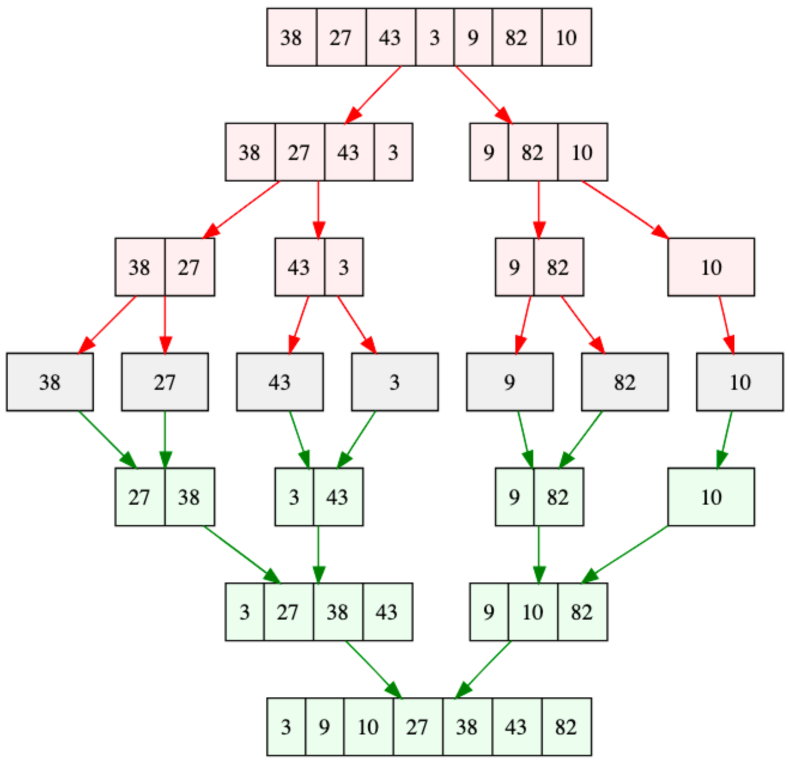
LA NOTION D'INVARIANT

INVARIANT

- Dans notre algorithme de tri par insertion, l'invariant de boucle est "Le tableau a[1:i] est trié" :
  - INITIALISATION : La valeur avant de rentrer dans la boucle est i=1, donc le tableau a[1:1] contient un seul élément. Un tableau contenant un seul élément est forcément trié (trivial), notre invariant "le tableau a[1:i] est trié" est donc vrai.
  - CONSERVATION : si l'invariant de boucle est vrai avant une itération de la boucle : "Le tableau a[1:i-1] est trié"
    Si on insère a[i] à sa place dans le tableau a[1:i-1], alors le tableau a[1:i] sera évidemment trié.
  - TERMINAISON : La dernière valeur prise de i dans la boucle est i=n, donc le tableau a[1:n] sera trié.

INVARIANT

- L'invariant pour le tri fusion est que chaque sous-liste (ou sous-tableau) créée lors de la division récursive est triée.
  - INITIALISATION (Division) : Lorsque le tri fusion divise récursivement une liste en deux moitiés, l'invariant est maintenu car chaque sous-liste créée est triée, car une seule valeur est considérée comme triée en soi.
  - CONSERVATION (Fusion) : Lorsque les sous-listes triées sont fusionnées pour créer une liste globalement triée, l'invariant est à nouveau maintenu.
  - TERMINAISON : Une fois que toutes les divisions et fusions récursives sont terminées, la liste finale est entièrement triée, respectant ainsi l'invariant initial.



MERCI POUR VOTRE ATTENTION