Algo 21 - Stratégie Diviser pour Régner

Les stratégies vues en 1er NSI :
→ la force brute
→ la stratégie gloutonne

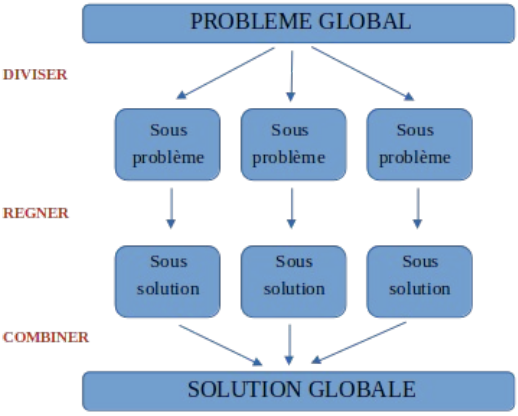
Les stratégies vues en Tle NSI
→ diviser pour régner
→ la programmation dynamique

I – Principe du diviser pour régner

1.1 Principe général du diviser pour régner

On découpe la résolution du problème en 3 phases :

- **Phase DIVISER** : division du problème global pour obtenir des petits problèmes plus faciles.
- **Phase REGNER** : on règne sur le sous-problème plus petit et qui est donc plus facilement gérable.
- **Phase COMBINER** : on combine les différentes sous-solutions pour obtenir la solution au problème global.



1.2 La différence avec la stratégie gloutonne ?

→ **Le sens de déroulement**

- La stratégie gloutonne est une stratégie HAUT-BAS qui réduit définitivement la taille du problème à chaque décision. C'est une stratégie HAUT → BAS
- DPR est une stratégie HAUT → BAS → HAUT à cause de la phase COMBINER.

→ **Le type de problème**

- La stratégie gloutonne vise à résoudre des problèmes d'optimisation, pour lesquels il existe **beaucoup de solutions** dont la qualité est variable.
- La stratégie diviser pour régner vise à résoudre des problèmes pour lequel il n'existe qu'**une solution**.

1.3 Exemple de stratégie diviser pour régner : la recherche DICHOTOMIQUE

→ Voir votre feuille de cours : ALGORITHME A NOTER ET A CONNAITRE PAR COEUR.

01° Pourquoi la recherche dichotomique fait partie des stratégies diviser pour régner plutôt que des stratégies gloutonnes alors qu'on fait bien un choix local à chaque étape ?

1.4 un autre exemple de stratégie diviser pour régner : rotation quart de tour d'une image carrée

→ Voir le cours en ligne

II – Le tri fusion, un tri utilisant la stratégie diviser pour régner

Le TRI PAR SELECTION :

→ **Principe** : **on récupère peu à peu les minimums dans le tableau et on inverse leurs positions avec l'élément 0 puis l'élément 1**… On peut faire la même chose en cherchant les maximums et en les plaçant dans la dernière case [-1] en Python, l'avant dernière [-2]…
→ **Coût** : **quadratique** dans tous les cas de figures. Pas de meilleur ou pire des cas.
→ **Complexité** : $\Theta(n^2)$

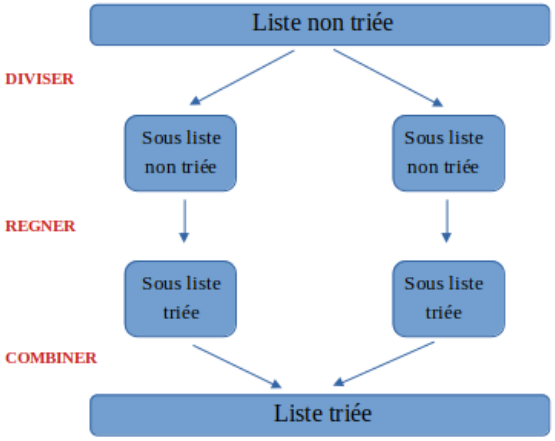
Remarque : **1 + 2 + 3 + … + n-1 + n =**

Le TRI PAR INSERTION :

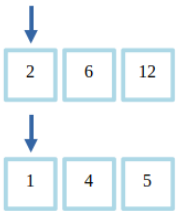
→ **Principe** : **on récupère un élément pour former un tableau trié. Ensuite, on ramène un élément à la fois et on le place correctement parmi les éléments déjà triés**.
→ **Coût** : **quadratique** dans le pire des cas mais **linéaire** dans le meilleur des cas.
→ **Complexité** : $\mathcal{O}(n^2)$

Remarque : cette fois, on ne réalise pas nécessairement tous les déplacements.

Principe général du tri fusion



- Phase **DIVISER :**
- Phase **REGNER** :
- Phase **COMBINER** :



Exemple de cours Réaliser sur papier le tri fusion du tableau `[5, 7, 3, 2, 6, 4, 1]`.

02° Réaliser sur papier le tri fusion du tableau `[45, 23, 7, 10, 50, 78, 12, 30]`

2.1 Coût de la phase DIVISER

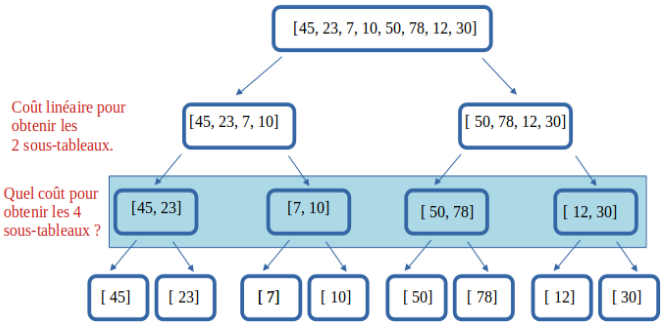
On notera **n** le **nombre de cases** du tableau.

03° Quel devrait-être le coût d'une division d'un tableau en deux en imaginant le prototype d'une fonction **`scinder()`** de ce type :
`scinder (t:tableau) -> (tableau, tableau)`

04° Le coût de la première division en deux du tableau est linéaire (en $\Theta(n)$). Quel devrait-être

- le nombre d'opérations nécessaires pour scinder le sous-tableau de gauche ?
- le nombre d'opérations nécessaires pour scinder le sous-tableau de droite ?
- le coût de l'opération totale sur cet étage de résolution du problème ?

05° Expliquer clairement pourquoi le coût de l'opération visant à diviser les 4 tableaux de l'étage du dessus est également linéaire par rapport au nombre d'éléments du tableau initial.



06° Quel est donc le nombre d'opérations nécessaires pour effectuer la division du tableau total en 8 petits tableaux ?

07° Quel est donc le nombre d'opérations nécessaires pour effectuer la division d'un tableau de 16 éléments en 16 petits tableaux ?

2.2 Coût de la phase REGNER

2.3 Coût de la phase COMBINER

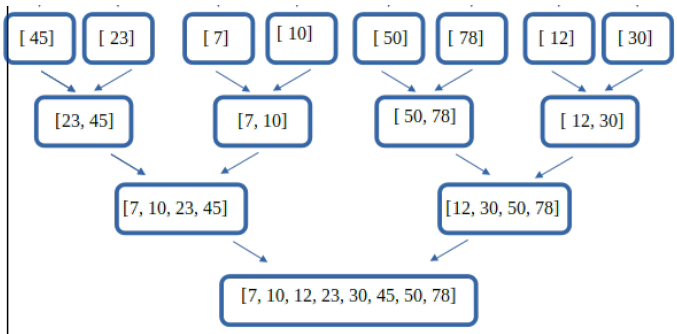
On considère un tableau A de n éléments et un tableau B de m éléments.

08° Quel va être le coût de la fusion des deux derniers sous-tableaux en un tableau trié en fonction de n, le nombre de cases du tableau initial ?

09° Quel va être le coût de la fusion des quatre derniers sous-tableaux en deux tableaux triés en fonction de n , le nombre de cases du tableau initial ?

10° Combien d'opérations au total pour fusionner les 8 sous-tableaux en un seul tableau trié ? Quel pourrait alors être le coût de cette opération ?

1. **Logarithmique** (complexité en $\log_2(n)$)
2. **Linéaire** (complexité en n)
3. **Quasi-linéaire** (complexité en $n \log_2(n)$)
4. **Quadratique** (complexité en n^2)



11° Combien d'opérations au total pour fusionner 16 sous-tableaux en un seul tableau trié ? Quel pourrait alors être le coût de cette opération ?

2.4 CONCLUSION : Coût du tri fusion sur un tableau

On peut décomposer le **tri fusion** en 3 étapes selon le principe de **diviser pour régner**.
Sur un tableau initial de n éléments, le coût des étapes est :

1. Phase **DIVISER : coût quasi-linéaire** (avec une complexité en $n \log_2(n)$)
2. Phase **REGNER : coût constant** (et même nul ici)
3. Phase **COMBINER : coût quasi-linéaire** (avec une complexité en $n \log_2(n)$)

Au total, le tri fusion d'un tableau est donc à _____

III – TP Prog : Le tri fusion avec Python

L'implémentation totale avec documentation est sur le site.

```
def trier(t):
    '''Renvoie un nouveau tableau trié par l'algorithme TRI FUSION
    :: param t(list) :: le tableau d'élément à trier, peut être vide
    :: return (list)  :: une copie triée du tableau

    '''
    longueur = len(t)

    if longueur < 2:            # CAS DE BASE : REGNER
        return [v for v in t]

    else:                       # CAS RECURSIF : SCINDER – TRIER - FUSIONNER
        gauche, droite = scinder(t)          # désempaquetage d'un 2-uplet
        return fusionner(trier(gauche), trier(droite))
```

```
def scinder(t):
    '''Renvoie un tuple composé du tableau de gauche et de droite obtenu en divisant t en 2
    :: param t(list)  :: le tableau NON VIDE d'éléments à diviser
    :: return (tuple) :: un couple (list, list) comme (gauche, droite)

    '''
    longueur = len(t)
    premier = 0                           # indice première case
    dernier = longueur - 1                # indice dernière case
    milieu = (premier + dernier) // 2     # indice de la case du mileu

    gauche = [t[i] for i in range(premier, milieu+1)]    # fin non inclus d'où le +1
    droite = [t[i] for i in range(milieu+1, dernier+1)]  # fin non inclus d'où le +1

    return (gauche, droite)  # on renvoie bien UNE unique réponse qui est un tuple



def fusionner(t1, t2):
    '''Renvoie un nouveau tableau trié en fusionnant les deux sous-tableaux triés NON VIDES
    :: param t1(list) : un sous-tableau TRIE
    :: param t2(list) : un sous-tableau TRIE
    :: return (list)  : un tableau trié

    '''
    # Etape 1 - Création d'un tableau t_fusion pour récupérer la fusion
    longueur = len(t1) + len(t2)                # nombre de cases du tableau de fusion
    t_fusion = [None for x in range(longueur)]  # tableau temporaire pour la fusion

    # Etape 2 - Initialisation des indices du tableau t_fusion et des deux sous-tableaux
    i = 0          # indice de la case dans laquelle on va insérer la prochaine valeur
    i1 = 0         # indice de la case à surveiller dans le sous-tableau 1
    f1 = len(t1) -1 # indice de la dernière case possible dans le sous-tableau 1
    i2 = 0         # indice de la case à surveiller dans le sous-tableau 2
    f2 = len(t2) -1 # indice de la dernière case possible dans le sous-tableau 2

    # Etape 3 - Remplissage t_fusion jusqu'à ce qu'un des deux tableaux soit lu
    while i1 <= f1 and i2 <= f2:
        if t1[i1] <= t2[i2]:
            t_fusion[i] = t1[i1]
            i = i + 1
            i1 = i1 + 1
        else:
            t_fusion[i] = t2[i2]
            i = i + 1
            i2 = i2 + 1

    # Etape 4 - Remplissage de t_fusion avec le sous-tableau restant
    while i1 <= f1:
            t_fusion[i] = t1[i1]
            i = i + 1
            i1 = i1 + 1
    while i2 <= f2:
            t_fusion[i] = t2[i2]
            i = i + 1
            i2 = i2 + 1

    # Etape 4 - On renvoie le tableau fusionné
    return t_fusion
```