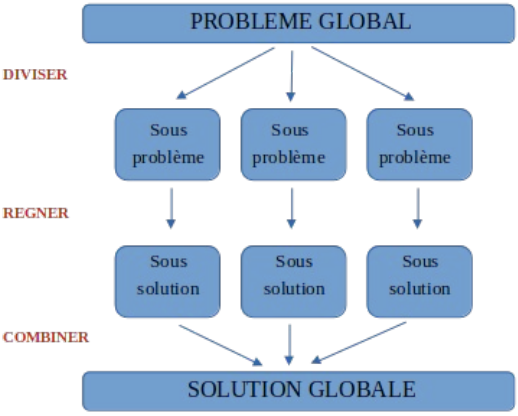
Algo 22 - Stratégie Diviser pour Régner

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** en plusieurs **sous problèmes indépendants** les uns des autres.
- **Phase REGNER** : on règne sur chaque petit sous-problème, plus petit donc plus facilement gérable.
- **Phase COMBINER** : on combine les **sous-solutions** pour obtenir la **solution globale** du problème initial.

1.2 La différence avec la stratégie gloutonne ?

→ **Le sens de déroulement**

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

→ **Le type de problème**

- La s**tratégie 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 cherche le minimum dans [0; n-1] et on l'inverse avec la case 0. On cherche ensuite le second minimum dans [1; n-1] et on l'inverse avec la case 1. On continue jusqu'à obtenir un tableau entièrement trié. On peut faire pareil en prenant le maximum et en le plaçant en fin de tableau.
→ **Coût** : **quadratique** dans tous les cas de figures. Pas de meilleur ou pire des cas.
→ **Complexité** : $\Theta(n^2)$

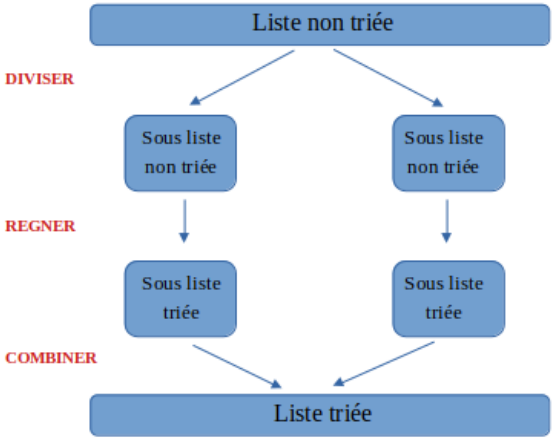
Remarque : **1 + 2 + 3 + … + n-1 + n =**

Le TRI PAR INSERTION :

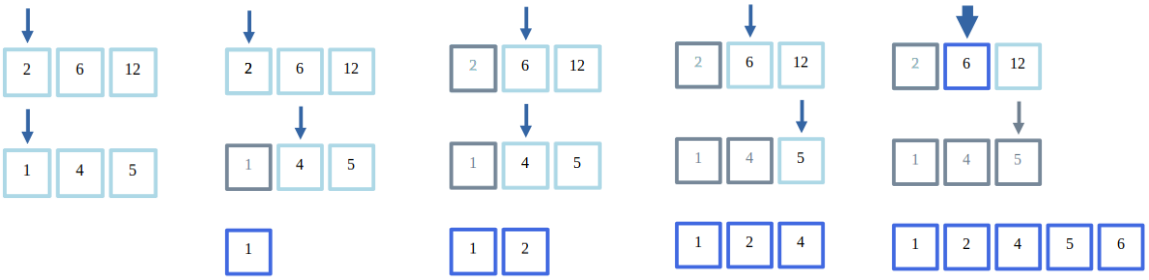
→ **Principe** : on crée un tableau d'un seul élément, puis on ramène et place correctement un deuxième élément. On continue jusqu'à ramener tous les éléments, un par un.
→ **Coût** : **quadratique** dans le pire des cas mais **linéaire** dans le meilleur des cas.
→ **Complexité** : $\mathcal{O}(n^2)$

Le tri par insertion est donc de meilleure qualité que le tri par sélection même si leur coût est similaire dans le pire des cas.

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.

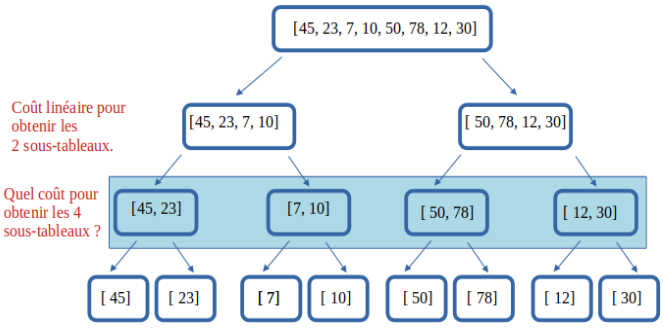
03° Quel devrait-être le coût T_1 pour obtenir les deux tableaux de l'étage 1 à partir du tableau de l'étage 0, en imaginant **`scinder()`** de ce type :

```
scinder (t:tableau) -> (tableau, tableau)
```

04° Quel devrait-être :

- le nombre d'opérations nécessaires pour scinder le sous-tableau de gauche comportant n/2 éléments ?
- le nombre d'opérations nécessaires pour scinder le sous-tableau de droite comportant n/2 ?
- le coût T_2 de l'opération totale sur cet étage ?

05° Quel va être le coût T_3 des opérations permettant d'obtenir le dernier étage à partir des tableaux de l'avant-dernier étage ?



06° Avec notre méthode de scinder en deux, quel est le nombre T_{total} d'opérations nécessaires pour scinder un tableau de 8 cases en 8 tableaux d'une case ?

07° Quel est le nombre T_{total} d'opérations nécessaires pour scinder un tableau de 16 cases en 16 tableaux d'une case ?

2.2 Coût de la phase REGNER

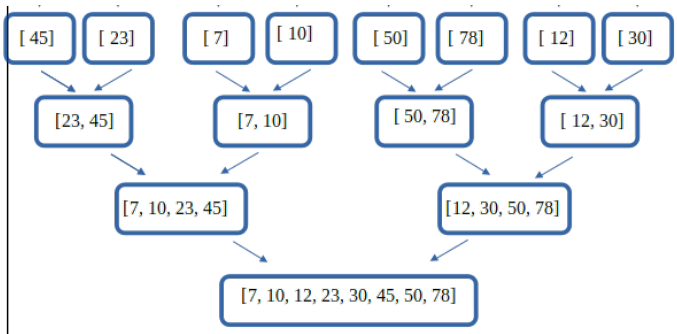
2.3 Coût de la phase COMBINER

08° Quel va être le coût T_3 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 T_2 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 est le coût total ?

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 ?

2.4 CONCLUSION : Coût du tri fusion sur un tableau

Lors de l'évaluation des trois phases DIVISER - REGNER - COMBINER, nous avons vu que le coût le plus important est un coût quasi-linéaire.

Le tri fusion est donc à coût quasi-linéaire, en $\Theta(n \log_2(n))$

III – TP Prog : Le tri fusion avec Python

```
def trier(t:list) -> list:
    '''Renvoie un nouveau tableau trié par l'algorithme TRI FUSION'''

    # CAS DE BASE
    if len(t) < 2:
        return [v for v in t]

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



def scinder(t :'list NON VIDE') -> (list, list):
    '''Renvoie un tuple composé des tableaux gauche et de droite en divisant t en 2'''

    premier = 0                                        # indice première case
    dernier = len(t) - 1                               # indice dernière case
    milieu = (premier + dernier) // 2                  # indice de la case du milieu

    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 UNE réponse qui est un tuple
```

```python
def fusionner(t1:'list TRIEE', t2:'list TRIEE') -> :'list TRIEE':
    '''Renvoie un tableau trié en fusionnant les deux sous-tableaux triés NON VIDES'''

    # Etape 1 - Création d'un tableau t_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
```