Diviser pour régner

Capacités attendues

- ✓ Écrire un algorithme utilisant la méthode « diviser pour régner » ;
- ✓ L'exemple du tri fusion permet également d'exploiter la récursivité et d'exhiber un algorithme de coût en $n\,log_2 n$ dans le pire des cas.

1 Notions d'algorithmique

Algorithme

Un algorithme est une suite d'instructions permettant de résoudre un problème.

Complexité d'un algorithme

La complexité d'un algorithme évalue la quantité de ressources (soit en temps, soit en mémoire) nécessaire à son exécution.

Complexité en temps dans le pire des cas
Dans le programme de NSI, on s'intéresse à la complexité en temps dans le pire des cas. On évalue alors le nombre d'opérations élémentaires exécutées, en fonction de la taille des données d'entrée notée n dans le pire des cas d'exécution possible. Par exemple, on recherche un élément qui n'est pas dans un tableau, il faut alors parourir tout le tableau, ce qui est *pire*, car plus coûteux en temps, que de le trouver en premier (si c'est le premier élément du tableau).

À quoi sert l'évaluation de cette complexité ?
La complexité dans le pire des cas permet de comparer l'efficacité de deux algorithmes résolvant le même problème.

Exercice 1

Tracer les différents types de complexité que vous connaissez sur un graphique. Donner pour chacune un exemple d'algorithme associé.

2 Les algorithmes du type « diviser pour régner »

Diviser pour régner est une *méthode de conception algorithmique* (une manière de résoudre des problèmes algorithmiquement).

Son principe est de en équivalents, mais de taille plus réduite.

Ces *petits problèmes* seront plus simples à résoudre que le problème initial. Une fois résolus, il faut les recombiner pour obtenir une solution au problème initial.

Trois principes de base pour diviser pour régner

- : on divise le problème initial en sous-problèmes ;
- : on résoud les sous-problèmes ;
- : on combine les solutions des sous-problèmes, pour avoir la solution du problème initial.

Le principe de la récursivité étant similaire à celui de cette approche, on utilise souvent des algorithmes récursifs. Cela n'est pas obligatoire ! Une approche *diviser pour régner* peut aussi être itérative (avec des boucles).

→ Nous avons déjà travaillé sur un algorithme appartenant à la famille *diviser pour régner*. Lequel ?

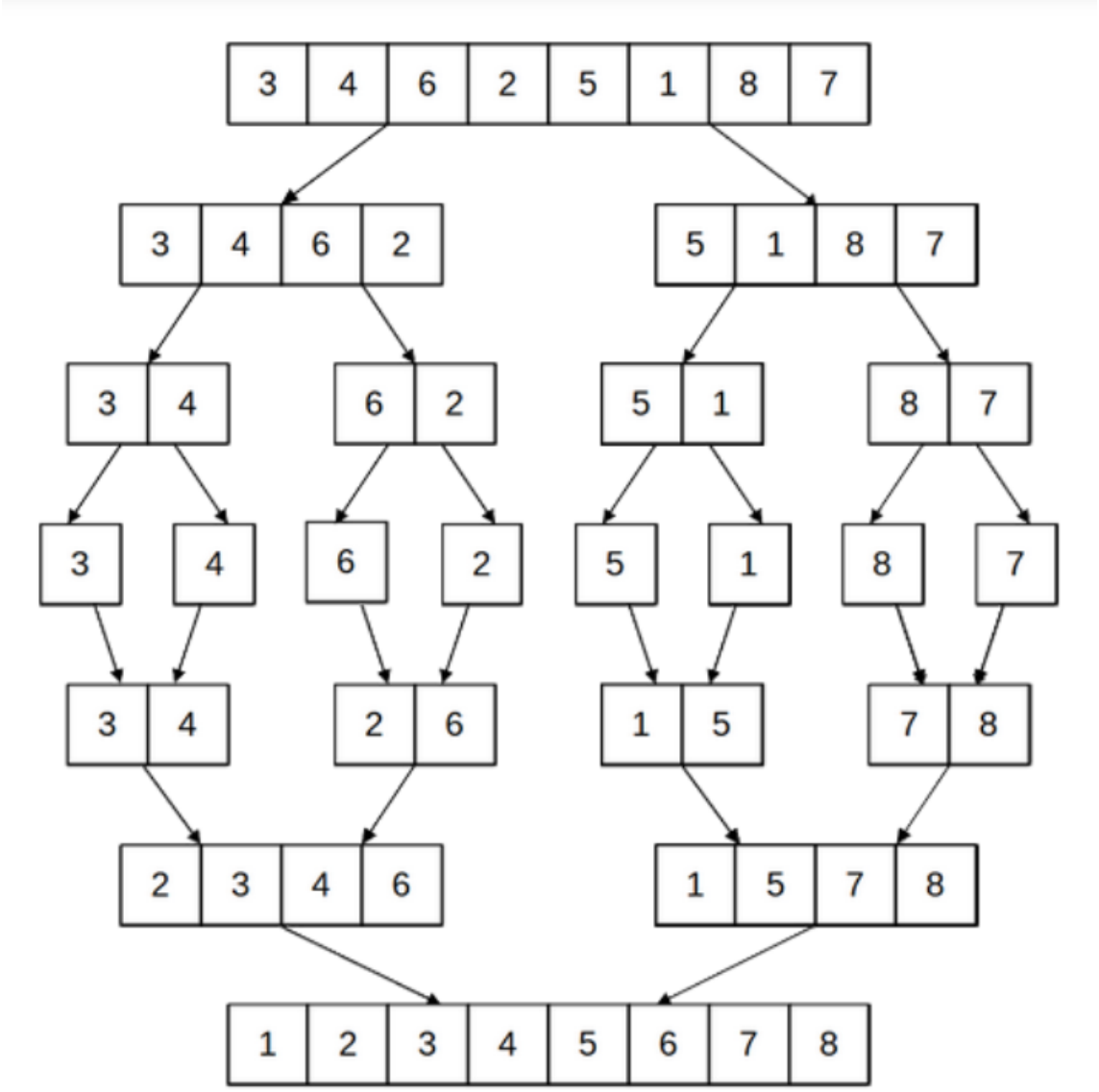
3 Le tri fusion

Principe et exemple

Le tri fusion est une manière de trier une liste en la divisant successivement, puis en fusionnant petit à petit les sous-listes triées pour finalement en obtenir une seule : la liste initiale triée.

Les trois étapes (selon les trois principes énoncés précédemment) de cet algorithme sont :

- Diviser : on divise le tableau successivement en deux, jusqu'à obtenir des tableaux de taille 1 ;
- Régner : on obtient un tableau de taille 1 qui donc est trié. À chaque nouvelle fusion, on s'assure que le résultat est trié.
- Combiner : on fusionne les tableaux triés deux à deux, jusqu'à en obtenir un seul.



Comparaison avec les autres algorithmes de tri

Tri par	Tri par
On sélectionne le minimum du tableau, et on l'échange avec l'élément en première position.	On prend le 2ème élément, on le compare avec le premier, et on échange les deux s'ils ne sont pas rangés dans l'ordre croissant.
On sélectionne le minimum de l'ensemble des éléments du tableau **sauf le premier**, et on l'échange avec celui en 2ème position.	On prend le 3ème élément, on le compare avec les précédents jusqu'à trouver un élément plus petit que lui ou arriver au début du tableau. On décale sur la droite les éléments plus grands, et on place l'élément 3 à la position trouvée.
On répète ces opérations, en réduisant de un en un la partie du tableau considérée, jusqu'à l'avant-dernier élément.	On répète ces opérations en parcourant de un en un les éléments du tableau, jusqu'au dernier.

Exercice 2

1. Quelle est la complexité de ces algorithmes ?

2. La complexité du tri fusion est en $n\ log_2\ n$. Est-il judicieux de l'utiliser à la place d'un tri par sélection ou par insertion ?

Exercice 3 – *(Les tris par insertion et par sélection : applications - révisions de 1ère -)*

1. Appliquer le tri par sélection aux tableaux suivants, en détaillant chacune de ses étapes :
   (a) `[9, 7, 1, 4, -2, 0, 8, 3]`
   (b) `[2, 6, 7, -2, 0, -5, 1]`
2. Appliquer ensuite le tri par insertion aux mêmes tableaux.

Exercice 4 – *(Les tris par insertion et par sélection : implémentation)*

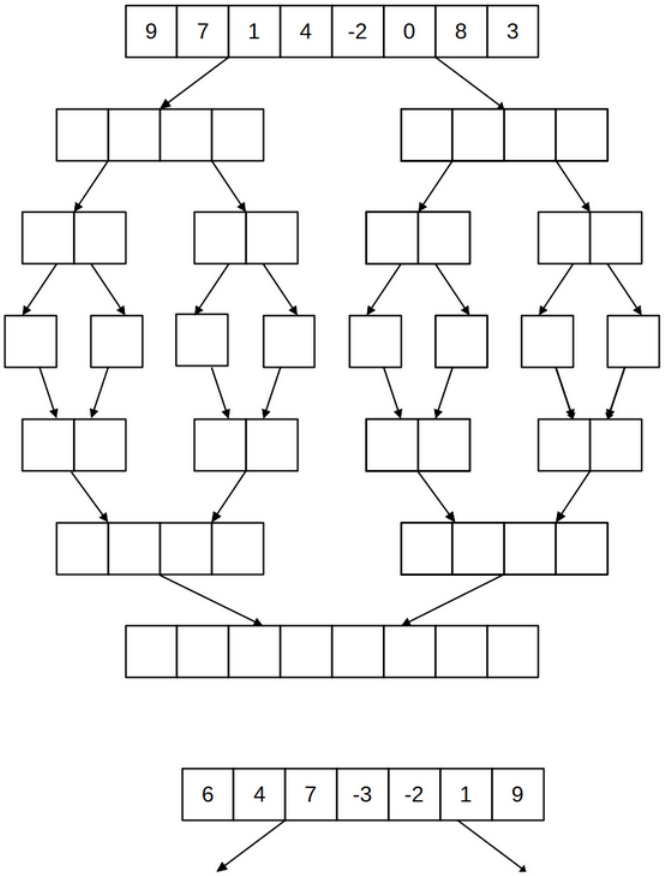
On se réfère au tableau de comparaison de ces algorithmes ci-dessus.

1. Pour chacun de ces algorithmes, identifier les opérations qui se répètent entre les étapes, et ce qui change. En déduire les opérations devant se trouver dans une boucle.
2. Formuler un pseudo-code (langage naturel) de chacun de ces algorithmes, reprenant les éléments de base de la programmation, mais sans la syntaxe Python.
3. Implémenter ces algorithmes dans deux fonctions `tri_selection(lst)` et `tri_insertion(lst)` prenant toutes deux en paramètre un tableau de type `list` lst en le modifiant **en place**.
4. Tester le fonctionnement de ces fonctions sur les deux exemples de l'exercice précédent.

Exercice 5 – *(Le tri fusion)*

1. Appliquer l'algorithme du tri fusion sur l'exemple ci-dessous, en complétant le schéma avec les divisions et recombinaisons successives.

   Dans le cas d'un tableau ayant un nombre impair d'éléments, on peut considérer que le tableau le plus petit va se retrouver à gauche, le plus grand à droite.



2. Il s'agit maintenant d'implémenter cet algorithme en Python. Nous allons suivre plusieurs étapes.

   L'implémentation du tri fusion se fait classiquement avec deux fonctions :

   - une fonction principale qui divise le tableau passé en paramètre par deux ;
   - une fonction qui fait la fusion entre deux tableaux triés.

(a) La fusion peut être écrite de manière récursive ou bien itérative. Après avoir complété et recopié le code ci-dessous dans votre éditeur, tester les deux versions sur les listes `[1,4,7,9]` et `[-2,0,3,8]`.

```
def fusion_it(lst1, lst2):
    taille1 = len(lst1)
    taille2 = len(lst2)
    lst_trie = []
    i1 = 0
    i2 = 0
    while (i1 < taille1) and (.............):
        if lst1[i1] < lst2[i2]:
            lst_trie.append(.............[i1])
            i1 = .............
        else:
            lst_trie.append(lst2[.............])
            i2 = .............
    while i1 < taille1:
        lst_trie.append(.............)
        i1 = .............
    while i2 < taille2:
        lst_trie.append(.............)
        .............
    return lst_trie

def fusion_rec(lst1,lst2):
    # premier cas de base, pour une valeur triviale de lst1
    if .............
        return lst2
    # deuxième cas de base, pour une valeur triviale de lst2
    elif .............
        return lst1
    elif lst1[0] < lst2[0]:
        # on place le bon élément en premier
        return [.............] + fusion_rec(............., .............)
    else:
        return [.............] + fusion_rec(............., .............)
```

(b) La fonction permettant de trier complètement le tableau divise d'abord en deux le tableau initial, avant de fusionner les résultats des tris sur la première, et la deuxième partie du tableau.

Compléter le code suivant :

```
def tri_fusion(lst):

    if ....................     # cas de base
        return lst
    else:
        # on coupe le tableau en deux sous-tableaux
        lst1 = [lst[i] for i in range(.............)]
        lst2 = [lst[i] for i in range(.............)]

    return fusion_rec(tri_fusion(.............), tri_fusion(.............))
```