

Tri d'une liste

Ambre Le Berre

MPSI

2025/2026

Plan du cours

1. C'est quoi un tri ?
2. Pourquoi trier ?
3. Comment trier ?

C'est quoi un tri ?

C'est quoi un tri ?

Trier une liste ou une collection d'éléments, ça consiste à les réordonner pour qu'ils soient en ordre croissant ou décroissant.

On peut écrire une condition nécessaire et suffisante pour qu'une liste $l_0, \dots, l_n$ soit triée dans l'ordre croissant :

$$\forall i \in \llbracket 0, n - 1 \rrbracket l_i \leq l_{i+1}$$

Fonction de tri

On va en général définir des fonctions de tri, qui prennent en argument une liste et la trie. On distingue deux sortes de fonction :

- les tris **in-place** qui modifient la liste originale en réordonnant les éléments, afin qu'elle soit triée
- les tris qui renvoient une nouvelle liste, triée, qui contient exactement les mêmes éléments que la liste originale.

Pourquoi trier ?

Raisons de présentation

Dans une table, une liste, ou d'autres données destinées à un affichage, il est souvent utile de trier les éléments. Imaginez un dictionnaire où les mots sont dans un ordre aléatoire ?

En informatique, on se retrouve souvent dans une situation où il est plus rapide de trier une liste avant de l'utiliser.

- Recherche d'éléments : à la manière d'un dictionnaire, il est beaucoup plus rapide de chercher un élément dans une liste triée, que dans une liste dans le désordre.
-
-

En informatique, on se retrouve souvent dans une situation où il est plus rapide de trier une liste avant de l'utiliser.

- Recherche d'éléments : à la manière d'un dictionnaire, il est beaucoup plus rapide de chercher un élément dans une liste triée, que dans une liste dans le désordre.
- Ordonnancement
-

En informatique, on se retrouve souvent dans une situation où il est plus rapide de trier une liste avant de l'utiliser.

- Recherche d'éléments : à la manière d'un dictionnaire, il est beaucoup plus rapide de chercher un élément dans une liste triée, que dans une liste dans le désordre.
- Ordonnancement
- Et bien plus ...

Comment trier ?

Le tri à bulle

<https://www.youtube.com/watch?v=Cq7SMsQBEUw>

Le tri à bulle

Idée : à chaque étape, on compare deux éléments adjacents. Si ils sont dans le bon ordre, on les laisse. Sinon, on les échange.

```
def tri_bulle(L):  
    """Trie la liste L "in place" avec l'algorithme du tri à  
    bulle"""  
    for i in range(len(L) - 1):  
        for j in range(len(L) - 1 - i):  
            if L[j] > L[j+1]:  
                L[j], L[j+1] = L[j+1], L[j]
```

Quelle est la complexité du tri à bulle ? Avec $n = \text{len}(L)$.

Autres tris ?

Réfléchissez à d'autres manières de trier une liste. Comment procédez-vous, naturellement, si on vous demande de trier la liste [4, 1, 2, 8, 3, 1, 9] ?

Tri par sélection

Idée : On prend d'abord le minimum de la liste, et on le place au début. Puis on répète sur le reste de la liste.

```
def tri_selection(L):  
    """Trie la liste L "in place" avec l'algorithme du tri par  
    sélection"""  
    for i in range(len(L) - 1):  
        el_min = L[i]  
        for j in range(i + 1, len(L) - 1):  
            if L[j] < el_min:  
                el_min, L[j] = L[j], el_min  
        L[i] = el_min
```

Quelle est la complexité du tri par sélection ? Avec $n = \text{len}(L)$.

Tri par insertion

Idée : On insère, un par un, chaque élément à sa place dans une nouvelle liste triée

```
def insere(L_triee, x):  
    """Insère l'élément x à sa place dans la liste triée L_triee in-  
    place"""  
    for i in range(len(L)):  
        if L_triee[i] > x:  
            L_triee.insert(i, x)  
    return
```

Quelle est la complexité de la fonction insere ? Avec $n = \text{len}(L)$.

Tri par insertion

Idée : On insère, un par un, chaque élément à sa place dans une nouvelle liste triée

```
def tri_insertion(L):  
    """renvoie une nouvelle liste contenant les éléments de L, triée  
    par insertion."""  
    L_triee = []  
    for x in L:  
        insere(L_triee, x)  
    return L_triee
```

Quelle est la complexité de la fonction tri_insertion ? Avec $n = \text{len}(L)$.

Tri par insertion

```
def insere(L_triee, x):  
    for i in range(len(L)):  
        if L_triee[i] > x:  
            L_triee.insert(i, x)  
    return
```

```
def tri_insertion(L):  
    L_triee = []  
    for x in L:  
        insere(L_triee, x)  
    return L_triee
```

Pouvez-vous écrire une version in-place du tri par insertion ?

Tri par insertion in-place

On va faire une fonction `insere` qui prend en argument une liste `L` dont les `k` premiers éléments $(0, 1, \dots, k-1)$ sont triés dans l'ordre croissant, et un indice `k`.

La fonction insère alors l'élément `x` dans les `k` premiers éléments de `L`, en écrasant l'élément d'indice `k`.

Tri par insertion in-place

```
def insere(L, k, x):  
    #boucle sur (k-1), (k-2), ... , 1, 0  
    for i in range(k-1, -1, -1):  
        #Si x est plus grand que l'élément actuel, on le range juste  
après, et on s'arrête  
        if x > L[i]:  
            L[i+1] = x  
            return  
        #sinon, on décale l'élément d'un cran pour laisser de la  
place à x  
        else:  
            L[i+1] = L[i]  
    #si on arrive au bout, c'est que x est le plus petit élément  
    L[0] = x
```

Tri par insertion in-place

On utilise ensuite cette fonction pour réaliser le tri :

```
def tri_insertion(L):  
    """Trie L en ordre croissant, par insertion, in-place"""  
    #On insère le (k+1)ème élément dans les k premiers.  
    for k in range(len(L)):  
        insere(L, k, L[k])
```

Que se passe-t-il dans chacun des algorithmes si la liste est déjà triée ? Si elle est “presque” triée ?

Propriétés

Que se passe-t-il dans chacun des algorithmes si la liste est déjà triée ? Si elle est “presque” triée ?

En fait, si on regarde la fonction du tri par insertion **in-place**, en particulier l'insertion, elle ne fait qu'une seule itération si l'élément est à sa place. Donc tout se passe comme si la liste était simplement parcourue une fois, pour une complexité **dans le meilleur cas** en $O(n)$.