

algorithmes de tri

Term NSI — Octobre 2026

Algorithmes de tri par insertion et par selection. Revisions de NSI_1

1 Algorithmes de tri de classe de complexité quadratiques

Rappel: Un algo de tri a pour but de placer les éléments d'une liste dans l'ordre. Les tris présentés ci-dessous sont des tri EN PLACE, de classe de complexité quadratique.

1.1 Tri par insertion

1.1.1 Animation

Sur l'animation du tri par insertion, on voit que 1. Le tri par insertion est inefficace pour la liste triée à l'envers 2. le tri par selection met à peu près le même temps quel que soit la liste traitée. Et ce temps est toujours supérieur à celui du tri par insertion 3. les algorithmes de tri quadratiques sont très inefficaces par rapport aux tris modernes

1.1.2 Lecture de l'algorithme

- La méthode de tri par insertion est appliquée **pour chacun des éléments de la liste**, du rang 1 au rang $\text{len}(L)-1$: c'est le rôle de la boucle externe de choisir un nouvel élément à placer, à chaque nouvelle itération (chaque nouveau tour) => boucle `for`. La liste est supposée triée jusqu'à l'indice `i-1`.
- L'algorithme utilise une boucle interne `while` de variant `j` pour effectuer les déplacements: Pour un élément d'indice `i`, de valeur `temp`, on compare `temp` avec la valeur de chaque élément `i-1`, `i-2`, ... jusqu'à trouver un élément plus petit (ou parvenir au bord gauche, d'indice 0). On insère alors `temp` dans la partie triée, à sa place. La condition d'exécution est alors, avec le nouveau variant `j`: `while j>0 and L[j-1]>temp:`

Par exemple: `L = [8, 10, 11, 7, 14, 1, 2, 9]`, on a effectué le tri jusqu'à `i=2`, on passe à `i=3`, `temp` vaut 7.

- La liste est triée jusqu'au rang `i-1`. Sélectionner le premier élément non trié, `i`. Le réserver en laissant la place libre (**memoriser** `j = i` et `temp=L[i]`): `j=3`, `temp=7`.
- Tant que** `temp < L[j-1]`, faire glisser les éléments de la partie gauche (triée) vers la droite pour laisser la place libre à l'élément à insérer (correspond à un copier-coller `j-1 => j`). Cela nécessite 2 instructions: `L[j]=L[j-1]` et `i-=1`. On va jusqu'à `j=0` avec `temp = 7`
- insérer l'a carte l'élément réservé à sa place: `L[j]=temp`

```

1  def tri_insertion(L):
2      for i in range(len(L)):
3          temp = L[i]
4          j = i
5          while j>0 and L[j-1]>temp:
6              L[j]=L[j-1]
7              j -=1
8          L[j]=temp

```

Remarques: La partie triée va évoluer au fur et à mesure du tri. Avec `i=3`, on a `[7,8,10,11]` puis avec `i=4`, on a `[1,7,8,10,11]`, etc...

1.2 Tri par selection

1.2.1 Lecture de l'algorithme

- La méthode de tri par selection est appliquée **pour chacun des éléments de la liste**, du rang 0 au rang $\text{len}(L)-2$, l'avant dernier élément: c'est le rôle de la boucle externe de choisir un nouvel élément à placer, à chaque nouvelle itération (chaque nouveau tour) => boucle `for`. La liste est supposée triée jusqu'à l'indice `i-1`.
- Pour la boucle interne, on parcourt TOUS les éléments, de `i+1` jusqu'à `len(n)-1` => `for`
 - On *selectionne le premier élément* de la partie non triée, au rang `i` (on le *marque*).
 - On *observe* l'élément directement à sa droite
 - Si l'élément *observé* est *inférieur* à l'élément *marqué*, il devient le nouvel élément *marqué*.
 - Une fois arrivée au bout de la liste: on *permuté* l'élément `i` avec celui *marqué*.

Résumé:

Pour chaque élément de la liste, au rang `i`:

- rechercher l'indice du minimum `i_min` dans la partie de la liste `L[i:]`
- Puis permuter `L[i]` avec `L[i_min]`

```

1 def tri_selection(liste):
2     for i in range(len(liste)):
3         indice_min = i
4         for j in range(i + 1, len(liste)):
5             if liste[j] < liste[indice_min]:
6                 indice_min = j
7         liste[i], liste[indice_min] = liste[indice_min], liste[i]
8     return liste

```

Remarque: la partie triée ne change plus, une fois celle-ci établie. Les tours suivants ne vont pas modifier la position des éléments déjà triés. Par exemple, avec la liste `L = [8, 10, 11, 7, 14, 1, 2, 9]`, pour `i=2`, on obtient `L = [1,2,7,...]` puis pour `i =3`, on obtient `L = [1,2,7,8,...]`, etc...

1.3 Complexité algorithmique

1.3.1 Tri par insertion

On peut compter le nombre de déplacements / affectations réalisés pour trier les valeurs de la liste, en fonction de la valeur `i`. Pour simplifier, on compte `N` = le nombre d'opérations d'affectations dans la boucle interne, dans le pire des cas.

i	N
1	2
2	3
3	4
4	5
...	...
n-1	n

La somme `T(n)` de cette suite arithmétique est alors $T_n = (n-1) \times (n+2)/2$

Soit $O(n^2)$ pour la complexité asymptotique.

1.3.2 Complexité du tri par selection

Avec la même règle de calcul pour $T(n)$:

On obtient aussi $O(n^2)$ pour la complexité asymptotique.

1.4 Preuve

La propriété que l'on cherche à démontrer peut s'énoncer ainsi: $P(i)$: Si la liste est triée jusqu'au rang i , alors elle doit le rester au rang $i+1$.

2 Trier une table txt ou csv à partir d'une clé

2.1 Import d'une table

Dans chaque cas, on ouvre le fichier `f`, on crée un itérable en decoupant à partir des retours à la ligne. Puis on parcourt l'itérable et on place les lignes dans une liste.

Pour le TP1: dictionnaire de mots: `.txt`

```
1 with open('liste_francais.txt', encoding='utf-8') as f:
2     for mot in f.read().splitlines():
3         mots.append(mot)
```

Pour le TP2, classement UEFA: `.csv`

```
1 import csv
2 with open('datas/classement_uefa.csv', newline='') as csvfile:
3     spamreader = csv.reader(csvfile, delimiter = ";")
4     teams = []
5     for row in spamreader:
6         teams.append(row)
```

Ici, la fonction `csv.reader` avec le paramètre `;` va placer les éléments d'une ligne dans une liste. La structure va être celle d'un tableau.

2.2 Trier à partir d'une clé

1. Ajouter un nouveau paramètre `cle` aux fonctions `tri_insertion` et `tri_selection`.
2. Adapter les programmes des 2 fonctions pour trier par rapport à une clé. Modifier par exemple la condition dans la boucle `for` de la fonction de recherche du minimum (ttri par selection) avec `if float(T[k][cle])<float(T[indiceDuMin][cle]) :`
3. Trier maintenant la liste `teams` selon la colonne de rang 7.