# Algorithme de tri

Ce sont des algorithmes qui permettent d’organiser une collection d’objets grâce à une relation d’ordre, les différents objets faisant partie d’un même ensemble (ex : ensemble des réels à trier par ordre croissant/décroissant ; ensemble de mots à trier par ordre alphabétique…)

Au programme, uniquement le tri d’un tableau 1D (une liste) de nombre réels

Choix : Tri par valeurs croissantes.

## Tri par insertion :

### Analogie avec le tri d’un paquet de copies :

Soit un paquet de N copies notées. On suppose que les i 1re copies sont triées. On prend la (i+1)ème copie du paquet et on la fait descendre jusqu’à sa place dans le paquet. On s’arrête dès qu’on compare da note à une note inférieure ou égale à la sienne ou bien si la copie est arrivée en 1re position du paquet.

Si le paquet contient N copies suppose que la 1re du paquet non-trié est à priori la 1re du paquet trié, puis à partir de la 2ème copie, on effectue les opération décrites précédemment …

### Déroulement de l’algorithme sur un exemple :

Ex : Soit à trier. Donc initialement,

* On suppose tout d’abord que le 1er élément de la liste non-trié est le 1er élément de la liste triée. Cet élément est l’objet 10 affecté en L=0.
* 1re étape : déplacement (ou non) du 2ème élément, l’objet 7, affecté en L[1] ?

i = 1 🡪L[1]<L[0] ? Oui donc on va permuter L[1] et L[0] : l’objet 7 initialement affecté en L[0] se retrouve affecté en L[0] : son indice de position a varié au cours de cette étape ,il a diminué de 1. A ce stade L=[7,10,4,9,11,3]. On s’arrête car l’objet 7 est en 1re position.

* 2ème étape : déplacement (ou non) du 3ème élément, l’objet 4, affecté en L[2] ?

i = 2 🡪L[2]<L[1] ? Oui 🡪 permutation de L[2] et L[1]. A ce stade, L=[7,4,10,9,11,3]

🡪L[1]<L[0] ? Oui 🡪 permutation de L[1] et L[0]. A ce stade, L=[4,7,10,9,11,3]. On s’arrête car l’objet 4 est en première position.

* 3ème étape : déplacement (ou non) du 4ème élément, l’objet 9, affecté en L[3] ?

i =3 🡪L[3]<L[2] ? Oui 🡪 permutation de L[3] et L[2]. A ce stade, L=[4,7,9,10,11,3]🡪L[2]<L[1] ? Non 🡪On s’arrête.

* 4ème étape : déplacement (ou non) du 5ème élément, l’objet 11, affecté en L[4] ?

i = 4 🡪 L[4]<L[3] ? Non 🡪 On s’arrête.

* 5ème étape : déplacement (ou non) du 6ème élément, l’objet 3, affecté en L[5] ?

i = 5 🡪 L[5]<L[4] ? Oui 🡪 permutation L=[4,7,9,10,3,11]🡪L[4]<L[3] ? Oui 🡪 permutation L=[4,7,9,3,10,11]

…

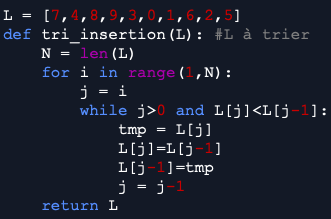
🡪L[1]<L[0] ? Oui 🡪 permutation L=[3,4,7,9,10,11]. ON s’arrête car l’objet j est en première position et le trie est fini car on a déplacé le dernier objet.

Bilan :

Pour trier par insertion une liste L contenant N réels, on procède :

* En N =1, étape analogue indexées par i, i variant de 1 à N-1.
* Au cours de la étape, on prend l’objet affecté en L[i] et on regarde si on doit se déplacer à partir de sa position initiale, on lui attribut alors un indice de position variable j pouvant varier de j=i à j=0 éventuellement par diminution de 1 à chaque fois.

### Algorithme :



### Amélioration de l’algorithme :

On reprend l’exemple précédent, à la 3ème étape i=3, L=[4,7,10,9,11,3]

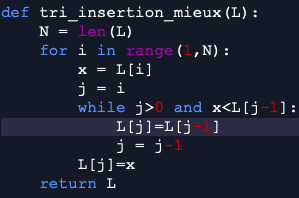
Déplacement (ou non) du 4ème élément de L : L[3] l’objet 9 ?

🡪 On garde en mémoire l’objet affecté en L[3], en l’affectant à une variable locale x : x=L[3].

Si x<L[3-1]=L[2] ? Oui 🡪 affectation de L[2] en L[3]. A ce stade, L=[4,7,10,10,11,3].

Si x<L[2-1]=L[1] ? Non donc x est à de L[1] donc en L[2] et l’étape est terminé.

A ce stade L = [4,7,9,10,11,3]. Globalement ce qui change c’es l’algorithme à l’étape i.



### Complexité temporelle dans le meilleur et dans le pire des cas

C’est le nombre d’opérations élémentaires que doit effectuer l’algorithme : affectation, comparaison, opérations arithmétiques. Ce nombre s’exprime en fonction du nombre de données N que l’algorithme, et son estimation nous donne celle du temps d’exécution. On dit que la complexité temporelle est O(n).

En général, on n’a pas besoin d’exprimer la complexité en comtant précis »ment lr nombre d’opérations élémentaire. Certaines etapes prenent bien moins dr temps que d’autre,on peut donc bien le rovet.

(A compléter)

Ex : Soit L=[10,7,4,9,11,3,15,22] à trier.

Len(L)>1 ? Oui 🡪 choix (arbitraire) d’un pivot. 10=p1 (c’est le premier pivot)

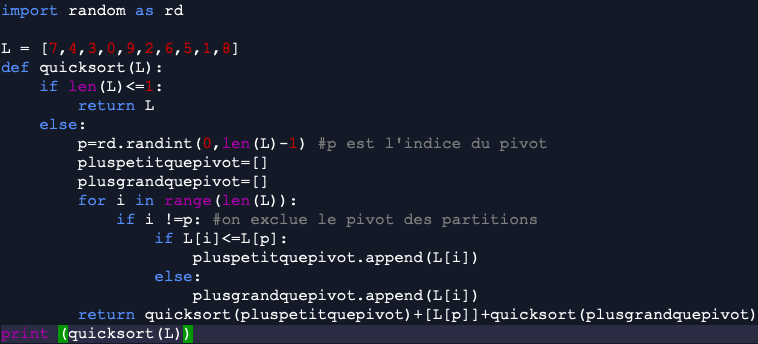
[7,4,9,3,…][10][11,15,22]…

Ensuite, il suffit de concaténer toutes les listes à 1 ou 0 éléments qui ont été renvoyées, de la gauche vers la droite : ici ça donne :

[3]+[4]+[7]+[9]+[]+[10]+[11]+[15]+[22], ce qui donne :

[3,4,7,9,10,11,15,22] : c’est L trié !

3) Code Python :

On rappelle que le module randint permet de générer « du hasard » et que la fonction randint(a,b) de ce module permet de générer un entier au hasard compris entre les entiers a et b inclus. 

Cette fonction est récursive à cause des appels récursifs à la dernière ligne.

4) Complexité temporelle :

Quand on appelle quicksort pour trier une liste à N-éléments, elle s’appelle elle-même (récursivité) pour trier une partition à k éléments et une autre partition à N-1-k-éléments. La valeur d k dépend du choix du pivot. Soit C la complexité temporelle de la fonction :

C(N)=N+C(k)+C(N-1-k)

C(1)=C(0)=0

On aboutit donc à une relation de récurrence qui dépend de k donc du choix du pivot.

1. Dans le pire des cas :

🡪 Si à chaque étape, un pivot est un extrémum de sa partition, on aura alors une sous-partition à 0 éléments et une autre de taille inférieure de 1 par rapport à la partition de départ.

🡪Ici : C(N)=N+C(0)+C(N-1)

or C(0)=0 d’après la relation de récurrence : C(N)= N+C(N-1) Etape 1

=N+N-1+C(N-2) Etape 2

=N+N-1+N-2+C(N-3) Etape 3

=N+N-1+N-2+N-3+C(N-4) Etape 4

…

=jN-1-2-3-…-(j-1)+C(N-j) Etape jf

On aura : jf\ N-jf=1🡪jf=N-1 (car on sait que C(1)=0)

Donc :

C(N)=(N-1)N-0,5\*(N-1)(N-2)+C(1) = O(N)🡪

1. Dans le milieu des cas :

Si chaque partitionnement produit toujours deux partitions de taille égale ou égale à 1 près (selon la parité de la liste qui a été partitionnée). Cela est possible si à chaque étape, on choisit un médian pour partitionner.

Un médian de liste de nombre est un nombre de cette liste qui est supérieur ou égale à la moitié de ces nombres et inférieur ou égale à l’autre moitié. C’est le nombre qui se trouve au milieu de la liste quand elle triée. Environ au milieu si la liste a une taille paire.

On a toujours :

Mais dans ce cas :

…

on aura :

or

En informatique, on préfère utiliser log que ln donc on écrira plutôt :

III) Trie fusion :

1. Principe :

On découpe la liste à trier en des sous-listes de taille égales ou égales à 1 près selon la parité de la liste. On répète le processus avec les sous-listes jusqu’à des sous-listes ne contenant qu’un seul élément qui sont donc triées.

Ex : [10,7,4,9,11,3,15,22]🡪[10][7][4][9][11][3][15][22]

On va fusionner les listes triées obtenues deux à deux en une seule liste triée. On répète le processus jusqu’à ce que la liste de départ soit triées.

Ex : une liste à un élément obtenu précédemment :

[7,10] [4,9] [3,11] [15,22] 🡪[4,7,9,10] [3,11,15,22]🡪[3,4,7,9,10,11,15,22]

1. Déroulement de l’algorithme sur un exemple :

D : découpage en deux sous-listes de tailles égales à 1 près.

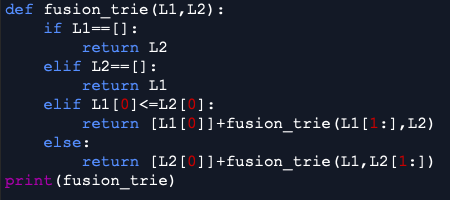
F : fusion de deux listes triées en une liste triée.

1. Implémentation de fonction récursives nécessaire au tri fusion :
2. Comment fusionner deux listes triées en une liste triée :

Ex : On veut fusionner triée avec triée. Le résultat doit être [4,7,8,9,10,12,13,22].

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Etape 1 | Etape 2 | Etape 3 | Etape 4 | Etape 5 | Etape 6 | Etape 7 |
|  |  | [8,12] | [Ø8,12] | [12] | [12] | [Ø12] | [] |
|  | [7,9,10,13,22] | [Ø7,9,10,13,22] | [9,10,13,22] | [Ø9,10,13,22] | [Ø10,13,22] | [13,22] | Ø[12,22] |
| Résultats | [4] | [4]+[7]  =[4,7] | [4,7]+[8]  =[4,7,8] | [4,7,8]+[9]  =[4,7,8,9] | [4,7,8,9]+[10]  [4,7,8,9,10] | [4,7,8,9,10]+[12]  =[4,7,8,9,10,12] | [4,7,8,9,10,12]+[13,22]  =[4,7,8,9,10,12,13,22] |

A chaque étape, on compare le premier élément de chaque liste triée, on retire le minimum que l’on insert dans une liste vide qui va être concaténée avec le résultat de l’étape précédent. Le processus s’arrête quand une de liste est vide. Il suffit alors de concaténer la liste restante avec le résultat de l’étape précédente



1. Comment effectuer le tri-fusion d’une liste L ?

On découpe la liste en deux sous-listes de longueur égale ou égale à un près (selon sa parité). On va appliquer le tri-fusion sur chacune des deux sous-listes crées à ‘étape précédente jusqu’à obtenir des listes de 1 élément trié et on fusionne le résultat avec la fonction fusion\_trie().

