TP d'informatique n° 12 : Méthodes de Tri

PCSI 2024 – 2025

Un algorithme de tri est un algorithme qui permet d'organiser une collection d'objets selon un ordre déterminé.
On s'intéresse ici à des méthodes de tri d'une liste de valeurs numériques. Celle-ci est implémentée sous la forme d'un tableau à une dimension. Les algorithmes de tri sont utilisés dans de très nombreuses situations. Ils sont en particulier utiles à de nombreux algorithmes plus complexes dont certains algorithmes de recherche, comme la recherche dichotomique.
La plupart des algorithmes de tri sont fondés sur des comparaisons successives entre les données pour déterminer la permutation correspondant à l'ordre croissant des données. Nous appellerons tri comparatif un tel tri.
La complexité de l'algorithme a alors le même ordre de grandeur que le nombre de comparaisons entre les données faites par l'algorithme.
Tous les algorithmes de ce TP sont des tris comparatifs.

Préambule :

Vous pourrez générer des listes non triées de taille de votre choix avec la fonction randint(a,b) du module random. Cette fonction un nombre aléatoire entre a et b et permet donc de créer des listes aléatoires avec un script comme celui-ci-dessous :

```
from random import randint
L=[randint(1,800) for N in range(500)]
```

On génère une liste aléatoire de 500 valeurs comprises entre 1 et 800.

A. Méthode de Tri Quadratique (« avec 2 boucles »)

I. Tri par sélection (comparatif)

Le premier tri que l'on veut implémenter s'appelle le tri par sélection. La méthode est la suivante : On cherche le plus petit nombre dans notre liste L et on le place au début d'un nouvelle liste res. On le supprime ensuite de L. On recommence ensuite : on recherche le nouveau plus petit nombre de L et on l'ajoute à la fin de res. On itère jusqu'à ce que L soit vide.

Exercice n°1 Écrire une fonction mini(L) qui prend en argument une liste de nombre et qui renvoie sous forme d'un tuple l'indice du minimum et la valeur du minimum. Par exemple mini ([0, -2,3]) doit renvoyer 1, -2

Exercice n°2 À l'aide de del et de la méthode append des listes, programmer une fonction tri_selection1(L) qui prend en argument une liste L et qui renvoie une liste qui contient les mêmes éléments que L mais dans l'ordre croissant (dit autrement qui trie L) en utilisant la méthode décrite ci-dessus.

Exercice n°3 Tester votre fonction sur une liste aléatoire que vous avez créé.

Rq : on parle de tri en place lorsque l'on fait le tri dans la liste à trier en échangeant les valeurs dans la liste.

Exercice n°4 Pour cet algorithme de tri par sélection le tri est-il fait en place ?

Il existe un algorithme de tri par sélection qui est dit en place.

Vous pouvez regarder une animation de ce tri sur ce lien :
http://lwh.free.fr/pages/algo/tri/tri_selection.html

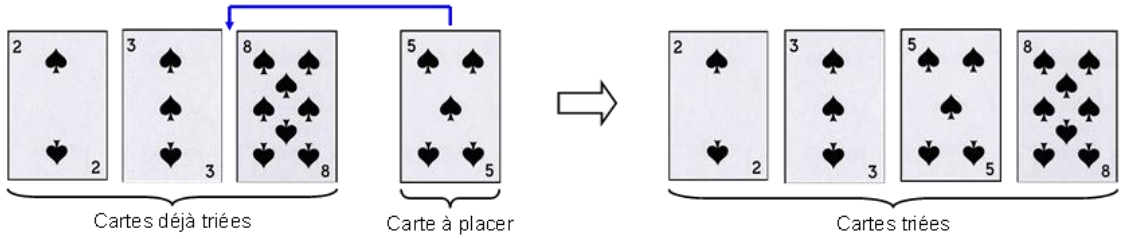
Une notion importante sur les algorithmes de tri est leur stabilité. Un tri est dit stable s'il préserve l'ordonnancement initial des éléments que l'ordre considéré comme égaux. Pour définir cette notion, il est nécessaire que la collection à trier soit ordonnancée d'une certaine manière (ce qui est souvent le cas pour beaucoup de structures de données, par exemple pour les listes ou les tableaux).

La stabilité de différents tri peut être testée en regardant ce site :
http://lwh.free.fr/pages/algo/tri/stabilite_tri.html

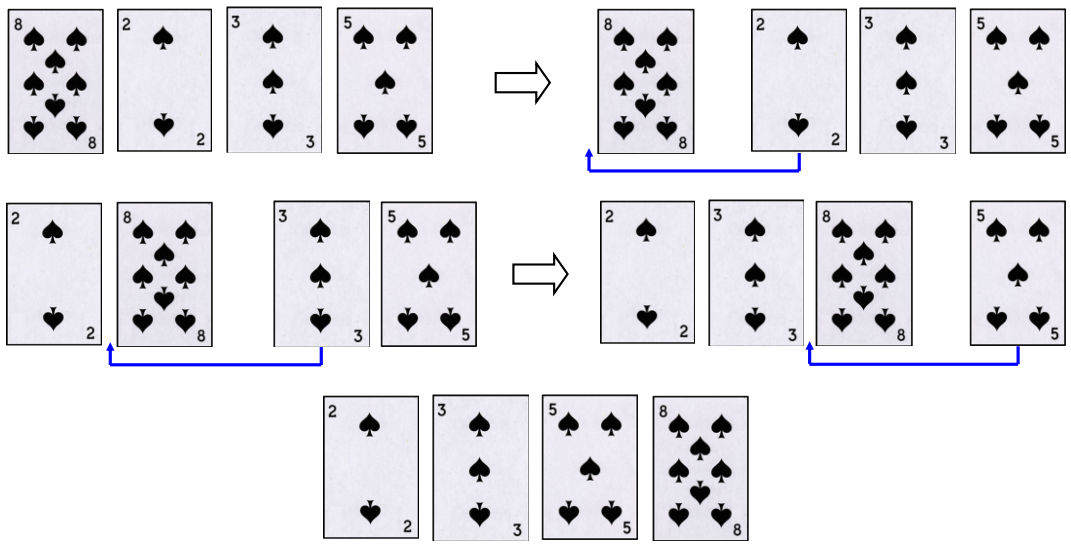
Exercice n°5 L'algorithme à tester sur le site est-il stable ? A votre avis qu'en est-il pour tri_selection1 ?

II. Tri par insertion (comparatif)

Le tri par insertion est considéré comme le plus naturel. Prenons l'exemple d'un joueur ayant déjà en main un 2, un 3, et un 8 de Pique, triés dans cet ordre. Si la quatrième carte qu'il tire est un 5 de pique, il l'insèrera naturellement entre le 3 et le 8.



Le procédé consiste à décaler d'une place vers la droite toutes les cartes déjà triées et ayant une valeur supérieure à celle de la nouvelle carte. Si l'ordre d'arrivée des 4 cartes est [8, 2, 3, 5], le tri complet (par insertion) peut se faire de la manière suivante :



Cela correspond à l'algorithme et à l'implémentation sous Python suivants :

Implémentation sous Python

```
def tri_insertion1(L) :
    for k,v in enumerate (L) :
        j=k
        while j>0 and v < L[ j-1 ] :
            L[ j ] = L[ j-1 ]
            j = j-1
        L[j]=v
    return L
# for i, Li in enumerate (L) donne simultanément
l'indice et la valeur de chaque élément de la liste.
```

Explications du programme en langage Python :

- *La boucle for permet de décrire la liste L à trier. v = L[k] représente la valeur de l'élément d'indice k, que l'on souhaite placer ;*
- *La boucle while permet de décaler d'un cran sur la droite les éléments déjà triés (L[j] = L[j-1]) avec j∈[0,k]) à condition que leur valeur soit supérieure à v (v < L[j-1]). La condition (j>0) permet d'arrêter la boucle while lorsque les éléments déjà triés ont tous une valeur supérieure à v ;*
- *L'affectation L[j] = v, permet d'installer la valeur v à sa place.*

Exercice n°6 Recopier ce script sous python et tester sur une liste aléatoire.

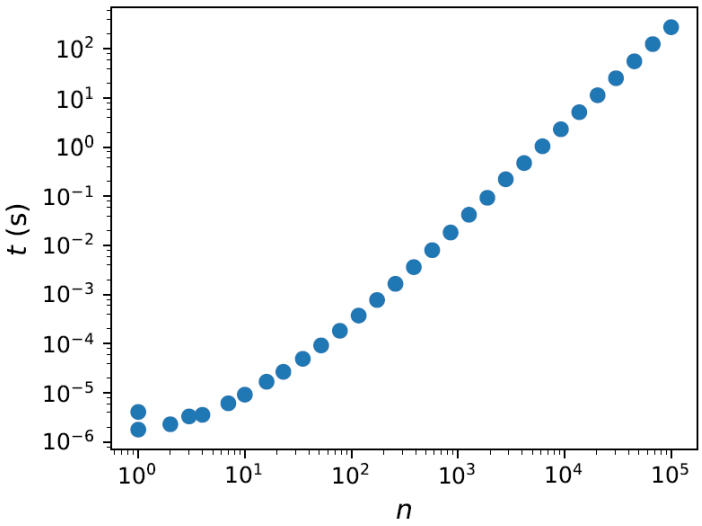
La stabilité de différents tri peut être testée en regardant ce site :
http://lwh.free.fr/pages/algo/tri/stabilite_tri.html

Exercice n°7 La fonction tri_insertion1 est-elle stable (tester sur le site) ?

III. Rapidité des algorithmes (complexité) [à traiter si vous êtes en avance]

La complexité des tris ci-dessus est en moyenne quadratique $O(n^2)$. Cette complexité est due aux deux boucles que réalisent les fonctions.

L'objectif est de mesurer expérimentalement la complexité de vos algorithmes de tri et ainsi d'obtenir une courbe de ce type :



Exercice n°8 Sur un tel graphique, montrer que si le temps d'exécution évolue comme une loi de puissance ($t = An^{\alpha}$ avec A et α des constantes), alors la courbe obtenue est une droite.

Exercice n°9 Ecrire un script python qui permet d'obtenir ce graphique pour vos fonctions tri_selection1 et tri_insertion1, valider la complexité de vos algorithmes

Le graphique étant semi-logarithmique vous devez utiliser la fonction loglog(x,y) de matplotib.

Pour avoir un tableau de plusieurs valeurs de n équirépartie sur une échelle logarithmique. Vous pouvez utiliser la fonction logspace de numpy qui prend pour argument puissance du premier élément, puissance du dernier élément et nombre d'élément. Par exemple si N = np.logspace (1, 3, 10), N est un tableau de 10 valeurs allant de 10^1 à 10^3 équirépartie de manière logarithmique. Pour le calcul du temps d'exécution vous pouvez vous inspirez du code ci-dessous :

```
from time import perf_counter
debut=perf_counter()
tri_insertion1(L1)
fin = perf_counter()
temps_execution =fin - debut
```

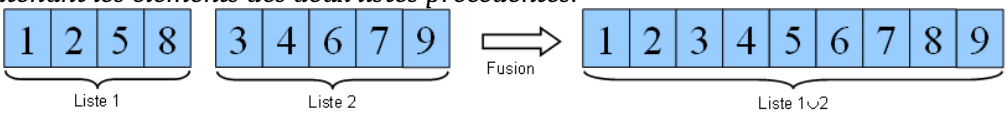
qui permet de calculer le temps d'exécution de la fonction tri_insertion.

B. Méthode de Tri « rapide » (avec ou sans récursivité)

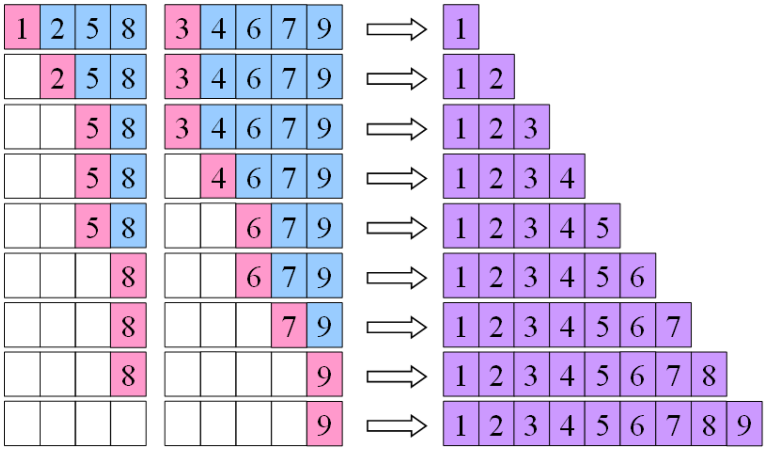
I. Tri par fusion (comparatif)

Etape 1 : Fusion de deux listes triées

Soient L1 et L2 deux listes déjà triées. Fusionner L1 et L2 revient à construire une nouvelle liste triée contenant les éléments des deux listes précédentes.



Comme les deux listes à fusionner sont déjà triées, la fusion s'opère en comparant seulement le premier élément de chacune. Le plus petit des deux est placé dans la liste fusionnée. Pour l'exemple précédent, cela donne :



Exercice n°10 Réaliser une fonction fusion(L1,L2) qui prend en argument deux listes que l'on suppose triées et qui renvoie une liste triée contenant tous les éléments de L1 et de L2. On prendra soin de réaliser une fonction de complexité O(len(L1) + len (L2)).

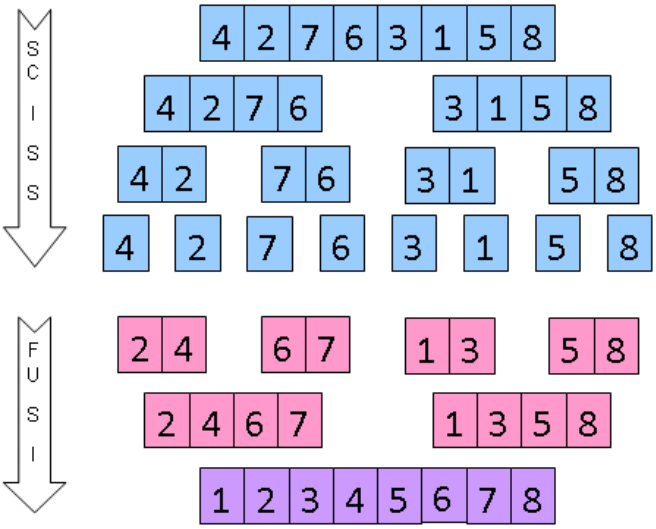
Proposition d'algorithme pour exercice 10 :

- *On fait évoluer deux indices i1 et i2 au fur et à mesure de l'avance dans L1 et L2 :*
  - *On initialise i1 et i2 à 0, et on crée une liste vide L ou du nombre d'éléments de L1 plus L2.*
  - *Tant que i1 < len (L1) et i2 < len (L2), comparer L1 [i1] et L2 [i2], ajouter la plus petite des deux à la fin de L et incrémenter l'indice correspondant (i1 ou i2)*
- *à la sortie de la boucle précédente, soit i1 = len (L1), soit i2 = len (L2) ; dans le premier cas, toutes les valeurs de L1 ont été recopiées dans L, il n'y a plus qu'à y recopier la fin de L2 ; idem dans le second cas.*

Par exemple : fusion([2, 5, 5,20], [5, 6,34]) => [2, 5, 5, 5, 6, 20, 34]

Etape 2 Tri par fusion

Maintenant que la fusion est définie, on revient au problème initial qui est de trier les éléments d'une liste. La liste originelle est scindée en deux parties de même taille, elles même partagées en deux autres parties de même taille, etc., jusqu'à ce que les sous listes n'aient plus qu'un élément. Ensuite il ne reste plus qu'à fusionner les sous listes.



La récursivité du mécanisme de scission apparaît naturellement. D'où l'implémentation du tri suivante :

Implémentation sous Python

```
def tri_fusion (L):
    if len(L) < 2: return L
    else:
        m=len(L)//2
        return fusion(tri_fusion(L[:m]),tri_fusion(L[m:]))
```

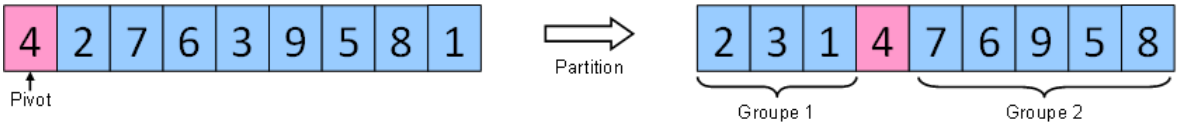
Exercice n°11 Recopier ce script sous python et tester sur une liste aléatoire.

La stabilité de différents tri peut être testée en regardant ce site :
http://lwh.free.fr/pages/algo/tri/stabilite_tri.html

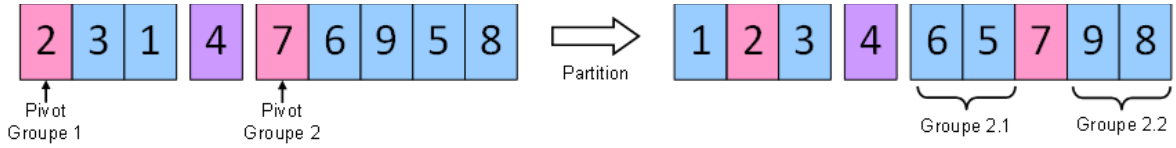
Exercice n°12 Le tri_fusion est-il stable (tester sur le site) ? Peut-on le qualifier d'en place ?

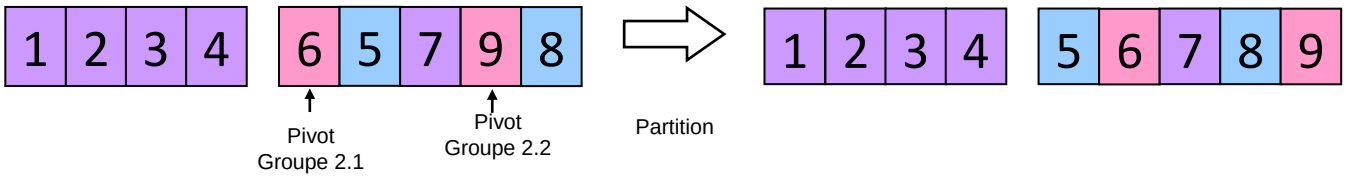
II. Tri rapide (comparatif)

Dans l'algorithme de tri rapide, A chaque appel de la fonction tri on choisit une valeur "pivot", par exemple le premier élément. On effectue une partition des éléments à trier. Un premier groupe est constitué de valeurs inférieures au pivot et un deuxième avec les valeurs supérieures. Le pivot est alors placé définitivement dans le tableau (la liste sous Python).



On traite alors chacun des groupes de façon indépendante. On peut les traiter avec le même algorithme (méthode récursive).





Cette méthode de tri peut être codé de manière récursive.

Exercice n°13 Réaliser une fonction tri_rapide(L) qui prend un argument une liste et qui renvoie une nouvelle liste triée. Votre fonction suivra l'algorithme suivant :

- *La première valeur de la liste est choisie comme pivot ;*
- *On crée deux listes G1 et G2 vides ;*
- *Si les valeurs de la liste L sont strictement inférieures au pivot, on ajoute cette valeur à G1 sinon on ajoute à G2 ;*
- *Une fois que ces deux Listes sont construits, on renvoie tri_rapide(G1)+[pivot]+tri_rapide(G2)*

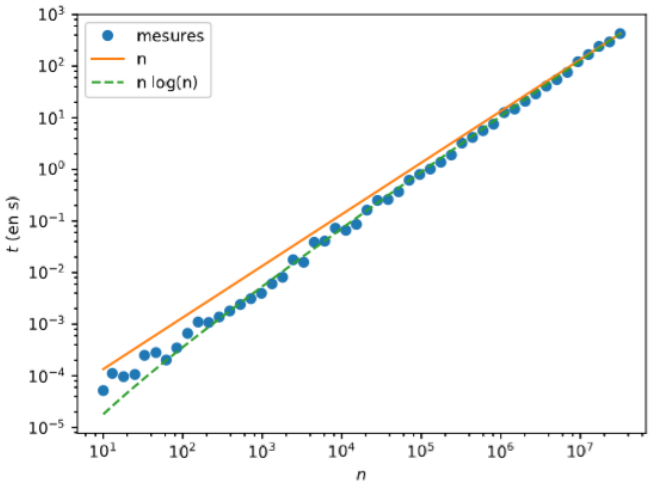
La stabilité de différents tri peut être testée en regardant ce site :
http://lwh.free.fr/pages/algo/tri/stabilite_tri.html

Exercice n°14 Votre fonction tri_rapide est-elle stable (tester sur le site) ? Peut-on la qualifier d'en place ?

III. Rapidité des algorithmes (complexité) [à traiter si vous êtes en avance]

La complexité des tris ci-dessus est en moyenne logarithmique O(log(n).n) . Cette complexité est due à la construction dichotomique des tri.

L'objectif est de mesurer expérimentalement la complexité de vos algorithmes de tri et ainsi d'obtenir une courbe de ce type :



Exercice n°15 Ecrire un script python qui permet d'obtenir un graphique (comme ci-contre) pour vos fonctions tri_fusion et tri_rapide, valider la complexité de vos algorithmes

C. Pour la prochaine fois : Méthode de Tri non comparatif : Tri par comptage

Le tri comptage (Counting sort en anglais), appelé aussi tri casier, est un algorithme de tri par dénombrement qui s'applique sur des valeurs entières. A la différence des précédents algorithmes de tri comparatifs, celui-ci n'utilise aucunes comparaisons, c'est son intérêt. Le principe repose sur la construction de l'histogramme des données, puis le balayage de celui-ci de façon croissante, afin de reconstruire les données triées. Ici, la notion de stabilité n'a pas réellement de sens, puisque l'histogramme factorise les données, plusieurs éléments identiques seront représentés par un unique élément quantifié Ce tri ne peut donc pas être appliqué sur des structures complexes, et il convient exclusivement aux données constituées de nombres entiers compris entre une borne min et une borne max connues.

L'algorithme présenté ici n'est pas la seule solution au problème mais elle permet de bien comprendre son fonctionnement :

- *Il est nécessaire de connaître la borne supérieure des valeurs entières de la liste à triée T. Il faut alors créer une liste de comptage, un tableau contenant n éléments, n'étant la valeur maximale dans la liste à triée, qui permet de compter les occurrences de chacune des valeurs dans la liste T. Pour chaque indice de T, on place ensuite les valeurs contenues dans la liste de comptage.*
- *On utilise à la liste de comptage pour renvoyer les valeurs de la liste T triée.*

Exercice n°16 Réaliser une fonction comptage(L,valeur_max) qui prend en argument une liste de nombre entier positif allant de 0 à une valeur maximum connue et la valeur maximum de la liste et qui renvoie une table de comptage.

Par exemple avec T= [2, 3, 4, 4, 5, 0, 1, 2, 3, 6, 5, 4, 1, 8, 9, 9, 7, 8, 8, 4, 6, 4, 5, 2, 3, 4] votre fonction doit renvoyer une table de comptage sous forme de liste :[1, 2, 3, 3, 6, 3, 2, 1, 3, 2]. Cette liste fournie l'information qu'il y a dans T : 1 zéro, 2 un, 3 deux, 3 trois, 6 quatre,.. et 2 neuf.

Exercice n°17 Réaliser une fonction tri_comptage(L,valeur_max) qui prend en argument une liste de nombre entier positif allant de 0 à une valeur maximum connue et la valeur maximum de la liste et qui renvoie une liste des éléments de L triés.

Par exemple avec T = [2, 3, 4, 4, 5, 0, 1, 2, 3, 6, 5, 4, 1, 8, 9, 9, 7, 8, 8, 4, 6, 4, 5, 2, 3, 4] votre fonction doit renvoyer : [0, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4, 4, 5, 5, 5, 6, 6, 7,8 ,8 ,8 ,9 ,9]