Algorithmes de tri

Tri à bulles

Preuve et complexité

Sommaire

I] Version brute du tri à bulles

1) Présentation de l'algorithme

Soit une liste quelconque d'entiers t, de taille N, qui est à trier dans l'ordre croissant.

La première étape du tri à bulles consiste à faire remonter le plus grand élément de la liste en dernière position de la liste, par comparaisons successives de tous les éléments consécutifs et éventuel échange de ces éléments.

Ensuite, le dernier élément étant le plus grand, on réitère le procédé à la liste des $N-1$ premiers éléments.

On réitère le procédé jusqu'à ce qu'il n'y ait qu'une comparaison (on trie finalement une liste à 2 éléments).

Cette première version du tri à bulles, qui sera étudiée dans cette partie I, est dite « brute » car elle continue à opérer des tests sur les éléments de la liste même si cela est devenu inutile, alors qu'il est possible d'optimiser le temps d'exécution avec un test d'arrêt supplémentaire : Cette amélioration sera développée dans la partie II.

Voici l'algorithme écrit sous forme condensé :

On se donne une liste d'entiers t, de taille N, indicée de 0 à $N-1$.

Pour i allant de 0 à $N-2$ faire :

    Pour j allant de 0 à $N-i-2$ faire :

        Si $t[j] > t[j+1]$ alors échanger les valeurs de $t[j]$ et $t[j+1]$

    Fin du pour

Fin du pour

2) Preuve de l'algorithme

Terminaison :

L'algorithme est composé de deux boucles de type « Pour » imbriquées, de pas 1, et aucune des variables i et j n'est impactée dans leurs exécutions. On peut donc majorer très grossièrement le nombre total d'itérations par $(N-1)^2$ (la boucle principale a $N-1$ itérations et le nombre d'itérations de la boucle interne est majoré grossièrement par $N-1$). Ce qui prouve la terminaison de l'algorithme.

Semi-preuve partielle :

Soit une liste d'entiers t, de taille N, indicée de 0 à $N-1$. Démontrons que la liste t est triée dans l'ordre croissant à la fin de l'algorithme.

Dans tout ce paragraphe, si a et b sont des entiers, on notera $[\![a;b]\!] = \{k \in \mathbb{N} \ / \ a \leq k \leq b\}$ (ensemble éventuellement vide).

Etude de la boucle interne :

On considère tout d'abord $i \in [\![0; N-2]\!]$ fixé, et on s'intéresse à la boucle interne, de compteur j :

Pour j allant de 0 à $N-i-2$ faire :

Si $t[j] > t[j+1]$ alors échanger les valeurs de $t[j]$ et $t[j+1]$

Fin du pour

Pour tout $j \in [\![0; N-i-1]\!]$ notons $Q_i(j)$ la propriété :
« L'élément de rang j est majorant des éléments d'indices strictement inférieurs à j », c'est-à-dire :

$$Q_i(j) : \quad \text{« } \forall k \in [\![0; j-1]\!] \, ; t[k] \leq t[j] \text{ »}$$

Démontrons que $Q_i(j)$ est un invariant de la boucle.

A l'entrée de boucle : Pour $j = 0$, la propriété est évidemment vraie puisque $[\![0; j-1]\!]$ est vide.

Soit $j \in [\![0; N-i-2]\!]$.
Supposons que la propriété $Q_i(j)$ soit vraie au début de l'itération de rang j de la boucle :
$\forall k \in [\![0; j-1]\!] \, ; t[k] \leq t[j]$.

Si $t[j] > t[j+1]$ est faux alors on a : $\forall k \in [\![0 \, ; j-1]\!] \, ; \quad t[k] \leq t[j] \leq t[j+1]$
donc : $\forall k \in [\![0 \, ; j]\!] \, ; \quad t[k] \leq t[j+1]$

Si $t[j] > t[j+1]$ est vrai alors, avant échange, on a : $\forall k \in [\![0 \, ; j-1]\!] \, ; \; t[k] \leq t[j]$ et $t[j+1] \leq t[j]$
Après échange de $t[j]$ et $t[j+1]$, cela donne : $\forall k \in [\![0; j-1]\!] \, ; \; t[k] \leq t[j+1]$ et $t[j] \leq t[j+1]$.
et on a donc : $\forall k \in [\![0 \, ; j]\!] \, ; \quad t[k] \leq t[j+1]$.

Dans les deux cas, on a $\forall k \in [\![0 \, ; j]\!] \, ; \; t[k] \leq t[j+1]$ ce qui prouve que $Q_i(j+1)$ est vraie après l'itération de rang j de la boucle.

Ainsi la propriété $Q_i(j+1)$ sera vérifiée après chaque étape de boucle pour $j \in [\![0; N-i-2]\!]$, et en particulier $Q_i(N-i-1)$ est vérifiée en fin de boucle :

$$\forall k \in [\![0; N-i-2]\!] \; ; \; t[k] \leq t[N-i-1] \qquad (*)$$

Etude de la boucle principale :

On considère maintenant la boucle principale de compteur i :

Pour i allant de 0 à $N-2$ faire :

Pour j allant de 0 à $N-i-2$ faire :

Si $t[j] > t[j+1]$ alors échanger les valeurs de $t[j]$ et $t[j+1]$

Fin du pour

Fin du pour

Pour tout $i \in [\![0; N-1]\!]$, notons $P(i)$ la propriété :
« Les $i+1$ derniers éléments de la liste t sont triés dans l'ordre croissant et de valeurs supérieures à tous les éléments qui précèdent », c'est-à-dire :

$$P(i) : \quad \text{« } \forall (k,l) \in [\![N-i; N-1]\!]^2 \quad ; \quad k < l \implies t[k] \leq t[l] \qquad (\alpha)$$
$$\text{et } \forall (k,l) \in [\![0; N-1-i]\!] \times [\![N-i; N-1]\!] \quad ; \quad t[k] \leq t[l] \quad (\beta) \text{ »}$$

Démontrons que $P(i)$ est un invariant de la boucle.

A l'entrée de boucle : Pour $i = 0$, $P(i)$ est évidemment vraie puisque l'ensemble : $[\![N-i; N-1]\!]$ est vide.

Soit $i \in [\![0; N-2]\!]$.
Supposons que la propriété $P(i)$ soit vraie au début de l'itération de rang i de la boucle.

La boucle interne sur j effectue uniquement une permutation des valeurs d'indices $j \in [\![0; N-i-1]\!]$, donc les valeurs de t de rang dans $[\![N-i; N-1]\!]$ ne sont pas impactées, et celles de rang dans $[\![0; N-1-i]\!]$ sont globalement conservées.
Il s'ensuit que les propriétés (α) et (β) de $P(i)$ seront vraies à la fin de cette itération sur i :
$(\alpha) : \forall (k,l) \in [\![N-i; N-1]\!]^2 \quad ; \quad k < l \implies t[k] \leq t[l]$
$(\beta) : \forall (k,l) \in [\![0; N-1-i]\!] \times [\![N-i; N-1]\!] \quad ; \quad t[k] \leq t[l]$

De plus, on a démontré que $Q_i(N-i-1)$ est également vérifiée :
(*) : $\forall k \in [\![0; N-i-2]\!] \quad ; \quad t[k] \leq t[N-i-1]$

Vérifions que $P(i+1)$ est vérifiée :

Soit $(k,l) \in [\![N-1-i; N-1]\!]^2$ avec $k < l$;
Si $k \geq N-i$ alors (α) assure bien que $t[k] \leq t[l]$.
Si $k = N-1-i$ alors (β) assure que $t[k] \leq t[l]$.

Soit $(k,l) \in [\![0; N-2-i]\!] \times [\![N-1-i; N-1]\!]$;
(*) assure que $t[k] \leq t[N-i-1]$ et (β) assure que $t[N-i-1] \leq t[l]$, donc on a bien $t[k] \leq t[l]$

Finalement, $P(i+1)$ est vérifiée à la fin de l'itération de rang i de la boucle principale, et en particulier $P(N-1)$ sera vérifiée à la fin de cette boucle, ce qui donne :

$\forall (k,l) \in [\![1; N-1]\!]^2 ; k < l \implies t[k] \leq t[l]$ et $\forall (k,l) \in [\![0; 0]\!] \times [\![1; N-1]\!]; t[k] \leq t[l]$

soit finalement : $\forall (k,l) \in [\![0; N-1]\!]^2 ; k < l \implies t[k] \leq t[l]$.

La liste t est donc bien triée à la fin de l'algorithme.

3) Complexité en temps

Notons N la taille de la liste t.

Le nombre d'itérations de la boucle principale est $N-1$ et le nombre d'itérations de la boucle intérieure est $N-i-1$ où i est le compteur de la boucle principale.

Ainsi, le nombre d'itérations de la double boucle vaut :

$$\sum_{i=0}^{N-2}(N-i-1) = (N-1)+(N-2)+\cdots+1 = \frac{(N-1)(N-1+1)}{2} = \frac{N(N-1)}{2}$$

d'où on déduit $T(N) = \ominus(N^2)$: L'algorithme est de complexité quadratique.

Notons qu'ici, la complexité en moyenne, dans le meilleur et dans le pire des cas sont les mêmes puisque le nombre d'itérations est fixe et ne dépend pas de la liste initiale.

L'étude du cas où, par exemple, la liste est déjà triée permet de se rendre compte qu'une nette amélioration du tri est possible : Lorsque la liste est balayée par le tri et qu'aucun échange n'est effectué, on peut interrompre le tri et éviter ainsi des balayages ultérieurs inutiles. Et cette remarque vaut pour toutes les étapes d'un tri à bulles : Dès qu'un balayage n'a engendré aucun échange, le tri peut s'arrêter. Cette amélioration est proposée dans le paragraphe suivant…

II] Tri à bulles optimisé

1) Présentation de l'algorithme amélioré

Si lors d'une étape du tri à bulles, aucune modification n'est apportée à la liste, alors on peut arrêter son exécution prématurément. Ainsi, on peut proposer l'amélioration suivante, avec un booléen nommé NoModif :

On se donne une liste d'entiers t, de taille N, indicée de 0 à $N-1$.

Pour i allant de 0 à $N-2$ faire :

    NoModif ← True

    Pour j allant de 0 à $N-i-2$ faire :

        Si $t[j] > t[j+1]$ alors { échanger les valeurs de $t[j]$ et $t[j+1]$; NoModif ← False }

    Fin du pour

    Si NoModif vaut True alors sortir de la boucle (fin de l'algorithme)

Fin du pour

2) Preuve de l'algorithme

Terminaison :

On remarque que le nombre d'itérations de l'algorithme présenté ici est majoré par celui du tri à bulles brut étudié dans la partie I, ce qui prouve qu'il termine également en temps fini.

Semi-preuve partielle :

On reprend les notations de la partie I.2).

• Dans le cas où le test sur la valeur de NoModif n'est jamais vérifié, la boucle principale sera menée à son terme, et on a déjà démontré dans la partie I que, dans ce cas, la liste est bien triée.

• Supposons qu'il existe un rang i_0 de la boucle principale pour lequel le test sur NoModif soit vrai.
Cela signifie que la valeur de NoModif n'a pas été modifiée lors de l'exécution de la boucle interne de compteur j, et on en déduit donc que : $\forall j \in [\![0; N-i_0-2]\!]$; $t[j] > t[j+1]$ est faux

soit $\forall j \in [\![0; N-i_0-2]\!]$; $t[j] \leq t[j+1]$ $\quad (\gamma)$

Par ailleurs, on avait prouvé qu'à la fin de l'itération de rang i_0 de cette boucle principale, on a $P(i_0+1)$ qui est vérifiée, c'est-à-dire :

$\forall (k,l) \in [\![N-i_0-1; N-1]\!]^2$; $k < l \implies t[k] \leq t[l]$ $\quad (\alpha)$

et $\forall (k,l) \in [\![0; N-2-i]\!] \times [\![N-1-i_0; N-1]\!]$; $t[k] \leq t[l]$ $\quad (\beta)$

En combinant (γ) et (α), on obtient donc $\forall j \in [\![0; N-1]\!]$; $t[j] \leq t[j+1]$.

La liste t est donc bien triée à la fin de l'algorithme.

3) Complexité en temps

On reprend les notations de la partie I.3).

Meilleur des cas :

Dans le meilleur des cas, où la liste initiale est déjà triée, la boucle principale ne tournera que pour la valeur $i = 0$.
On aura donc $T_{\substack{meilleur \\ des\ cas}}(N) = N - 1$, d'où $\boxed{T_{\substack{meilleur \\ des\ cas}}(N) = \ominus(N)}$.

Pire des cas :

Dans le pire des cas, où le plus petit élément de la liste est situé en fin de liste, la double-boucle devra s'exécuter entièrement pour placer cet élément en tête de liste.
On aura donc $T_{\substack{pire\ des \\ cas}}(N) = \frac{N(N-1)}{2}$, d'où $\boxed{T_{\substack{pire\ des \\ cas}}(N) = \ominus(N^2)}$.

Complexité moyenne :

Considérons maintenant un échantillon composé de toutes les listes de taille N obtenues par permutations des éléments de $[\![1;N]\!]$. On considère ainsi que toutes les valeurs d'une liste sont distinctes, et on suppose que le choix d'un élément de cet échantillon (correspondant bijectivement à une permutation $\sigma \in S_N$) se fait de façon équiprobable.

Considérons la fonction ψ_N qui à chaque permutation $\sigma \in S_N$ associe le nombre d'inversions de cette permutation :

$$\begin{array}{rccl} \psi_N : & S_N & \to & \mathbb{N} \\ & \sigma & \mapsto & card\{(i,j) \in [\![1;N]\!]^2 / \ i<j \ ; \ \sigma(i)>\sigma(j)\} \end{array}$$

Lorsqu'une permutation $\tau_{j,j+1}$ est opérée dans la double boucle de l'algorithme sur un élément correspondant à une permutation σ, elle réordonne alors les deux éléments de σ de rangs j et $j+1$ donc :

$$\psi_N\left(\tau_{j,j+1} \circ \sigma\right) = \psi_N(\sigma) - 1$$

Il en résulte que le nombre de permutations opérées lors de l'exécution de l'algorithme sur une liste correspondant initialement à une permutation σ est égal à $\psi_N(\sigma)$. Le nombre d'itérations effectuées est donc grossièrement minoré par $\psi_N(\sigma)$.

Pour $(i,j) \in [\![1;N]\!]^2$, notons $X_{i,j}^{(N)}$ la variable aléatoire définie sur S_N par :

$$\forall \sigma \in S_N \ ; \ X_{i,j}^{(N)}(\sigma) = \begin{cases} 1 \ si \ \sigma(i) > \sigma(j) \\ 0 \quad\quad sinon \end{cases}$$

On a alors :

$$\psi_N(\sigma) = \sum_{\substack{1 \le i \le N \\ 1 \le j \le N \\ i<j}} X_{i,j}^{(N)}(\sigma)$$

(en considérant ψ_N comme une variable aléatoire sur S_N)

Par linéarité de l'espérance, on a donc :

$$E(\psi_N) = \sum_{\substack{1 \le i \le N \\ 1 \le j \le N \\ i<j}} E\left(X_{i,j}^{(N)}\right)$$

Comme chaque fonction $\begin{matrix} S_N \to & S_N \\ \sigma \mapsto & \tau_{i,j}\sigma \end{matrix}$ est clairement bijective et que (pour $i \neq j$), $X_{i,j}^{(N)}(\sigma) = 0 \Leftrightarrow X_{i,j}^{(N)}\left(\tau_{i,j}\sigma\right) = 1$, on en déduit que $p\left(X_{i,j}^{(N)} = 0\right) = p\left(X_{i,j}^{(N)} = 1\right) = \frac{1}{2}$, et donc finalement $E\left(X_{i,j}^{(N)}\right) = \frac{1}{2}$.

Finalement :

$$E(\psi_N) = \sum_{\substack{1\leq i\leq N \\ 1\leq j\leq N \\ i<j}} \frac{1}{2} \;=\; \frac{1}{2} \times card\{(i,j) \in [\![1;N]\!]^2 /\; i<j\} \;=\; \frac{1}{2} \times \frac{N^2-N}{2} \;=\; \frac{N(N-1)}{4}$$

Cette valeur est un minorant de la complexité moyenne du tri à bulles, qui peut donc être au mieux quadratique. Or la complexité dans le pire des cas, qui est un majorant de la complexité moyenne, est d'ordre quadratique.

Donc la complexité moyenne du tri à bulles est exactement d'ordre quadratique :

$$\boxed{T_{moyen}(N) = \ominus(N^2)}.$$

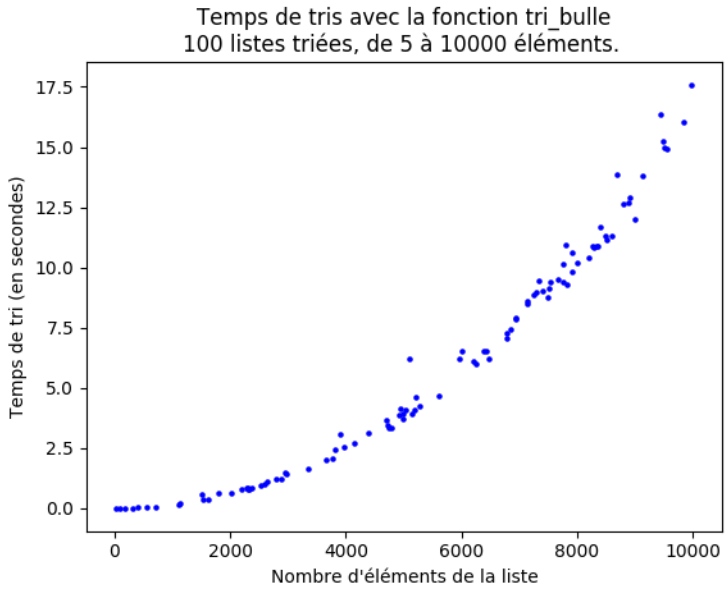
III] Implémentations en langage Python

1) Tris à bulles et vitesse d'exécution

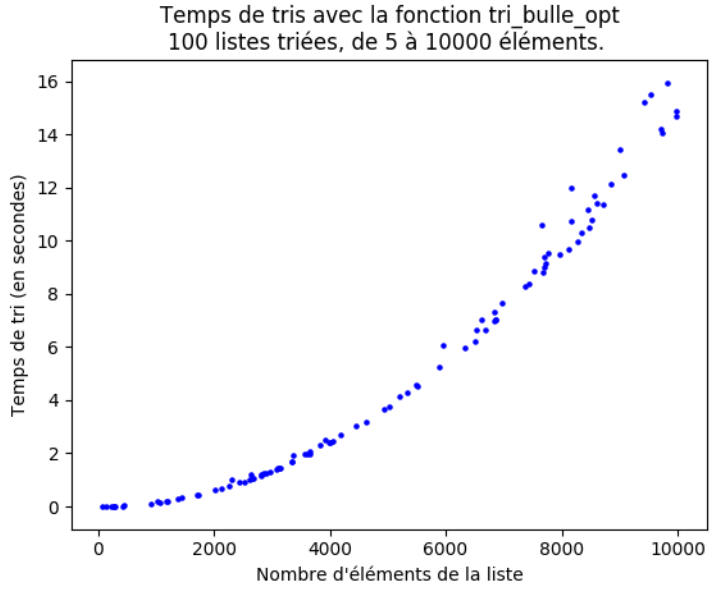
Les fonctions Python proposées ici ne renvoient aucune valeur, puisqu'elles agissent directement sur les listes Python pointées en argument.
La complexité en temps quadratique de ces fonctions peut être mise en évidence expérimentalement en mesurant les temps de tris pour des listes de tailles diverses, puisqu'on obtient ainsi des courbes d'aspects paraboliques. Bien sûr, les résultats numériques dépendent fortement du matériel.

```
def echange(t,i,j):
    """
    échange des éléments de rangs i et j de la liste t
    """
    t[i],t[j]=t[j],t[i]


def tri_bulle(t):
    """
    algorithme brut de tri à bulles
    """
    N=len(t)
    for i in range(N-1):
        for j in range(N-i-1):
            if t[j]>t[j+1]:
                echange(t,j,j+1)
```




```
def tri_bulle_opt(t):
    """
    algorithme de tri à bulles optimisé
    """
    N=len(t)
    for i in range(N-1):
        NoModif=True
        for j in range(N-i-1):
            if t[j]>t[j+1]:
                echange(t,j,j+1)
                NoModif=False
        if NoModif: break
```

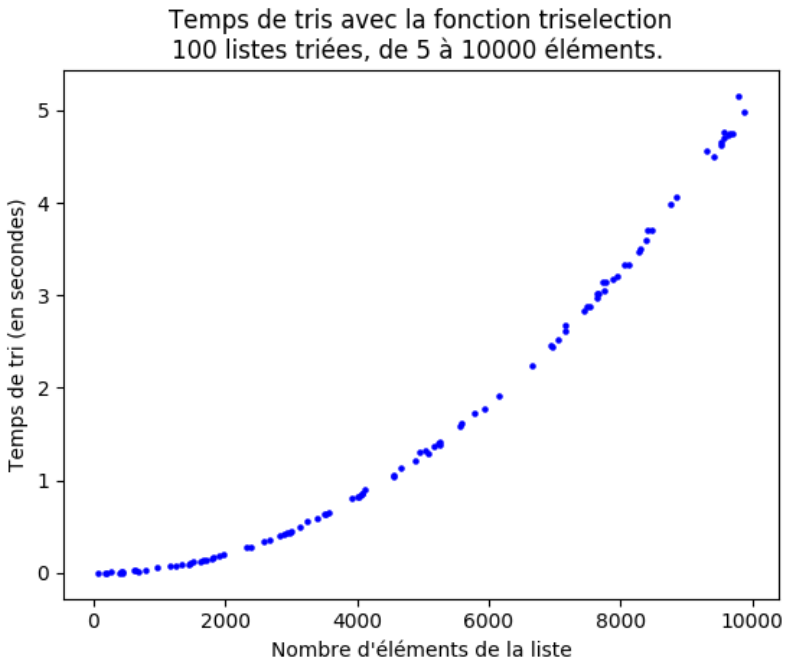


2) Éléments de comparaison avec d'autres tris

A titre de comparaison, voici ci-dessous quelques graphiques obtenus avec d'autres algorithmes de tris classiques. Hormis le tri fusion, les algorithmes ont été testés dans leur version explicite non récursive (on atteint dans la plupart des cas très rapidement les limites de profondeur de récursivité).

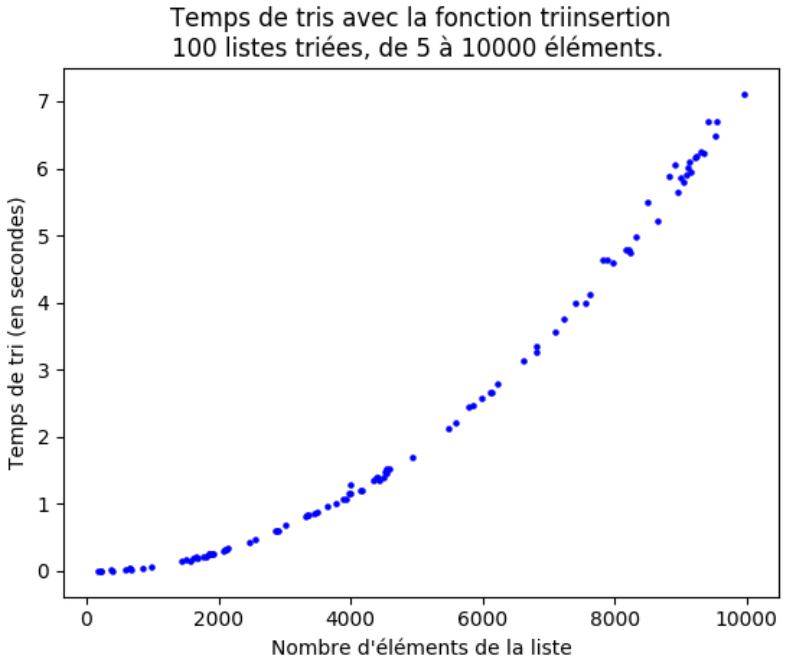
Tri par sélection (complexité moyenne : $\ominus(N^2)$)

```
def triselection(t):
    """
    algorithme direct de tri par sélection
    """
    for i in range(len(t)):
        k=i
        for j in range(i+1,len(t)):
            if t[j]<=t[k]:
                k=j
        echange(t,i,k)
```



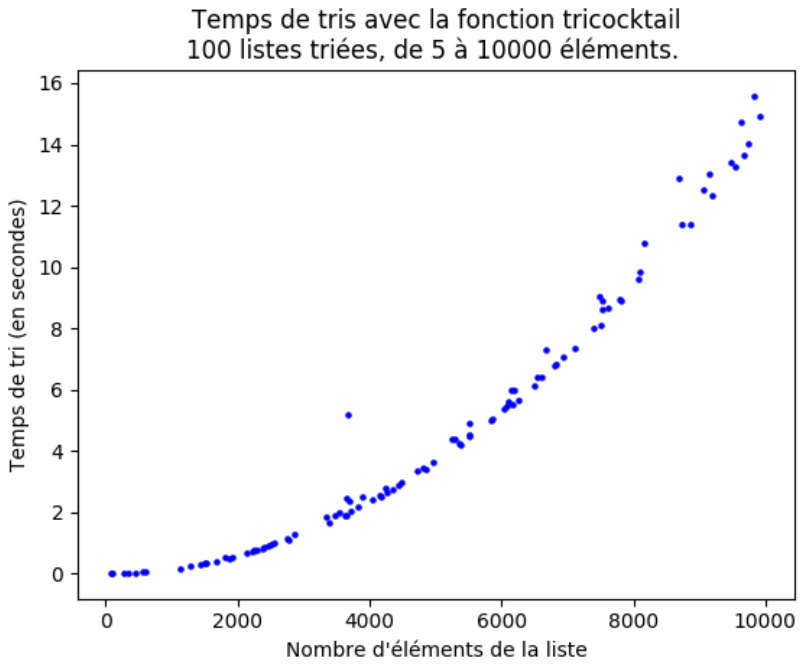
Tri par insertion (complexité moyenne : $\ominus(N^2)$)

```
def triinsertion(t):
    """
    algorithme direct de tri par insertion
    """
    for i in range(1,len(t)):
        ins=t[i]
        j=i
        while j>0 and ins<t[j-1]:
            t[j]=t[j-1]
            j-=1
        t[j]=ins
```



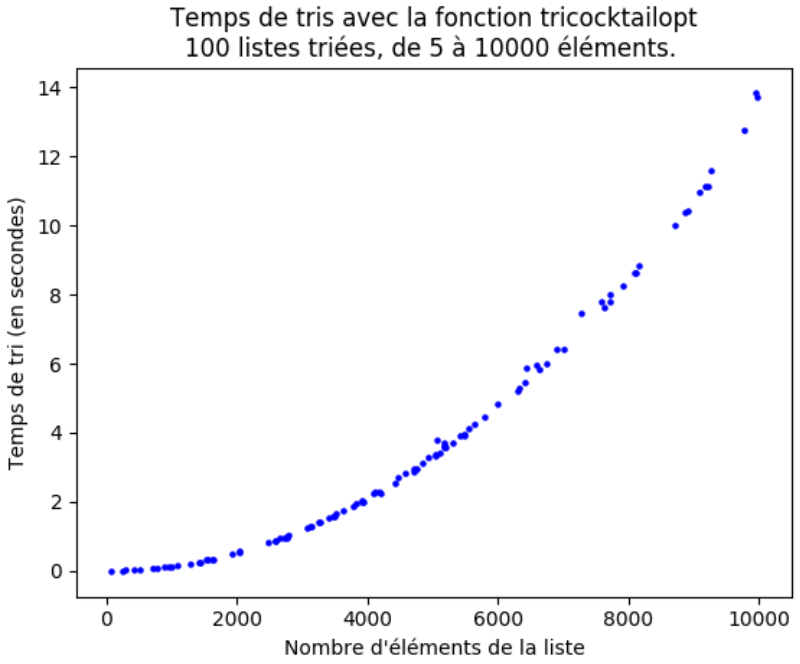
Tri cocktail (complexité moyenne : $\ominus(N^2)$)

```
def tricocktail(t):
    """
    algorithme direct de tri cocktail
    """
    for i in range(len(t)):
        for k in range(i,len(t)-i-1):
            if t[k]>t[k+1]:
                echange(t,k,k+1)
        for k in range(len(t)-i-1,i,-1):
            if t[k]<t[k-1]:
                echange(t,k,k-1)
```



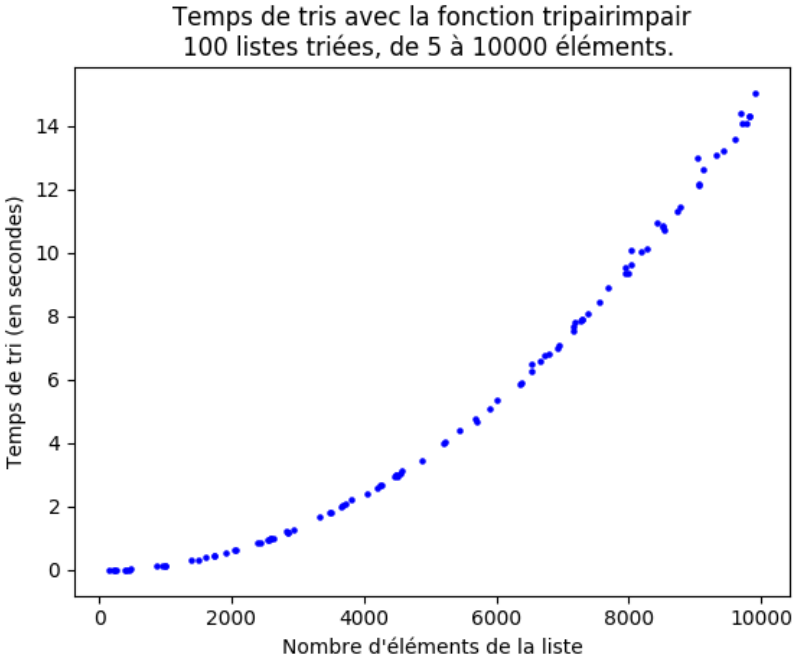
Tri cocktail optimisé (complexité moyenne : $\ominus(N^2)$)

```
def tricocktailopt(t):
    """
    algorithme de tri cocktail optimisé
    """
    TriOK=False
    début=0
    fin=len(t)-1
    while not TriOK:
        TriOK=True
        for i in range(début,fin):
            if t[i]>t[i+1]:
                echange(t,i,i+1)
                TriOK=False
        fin=fin-1
        for i in range(fin,début,-1):
            if t[i]<t[i-1]:
                echange(t,i,i-1)
                TriOK=False
        début=début+1
```



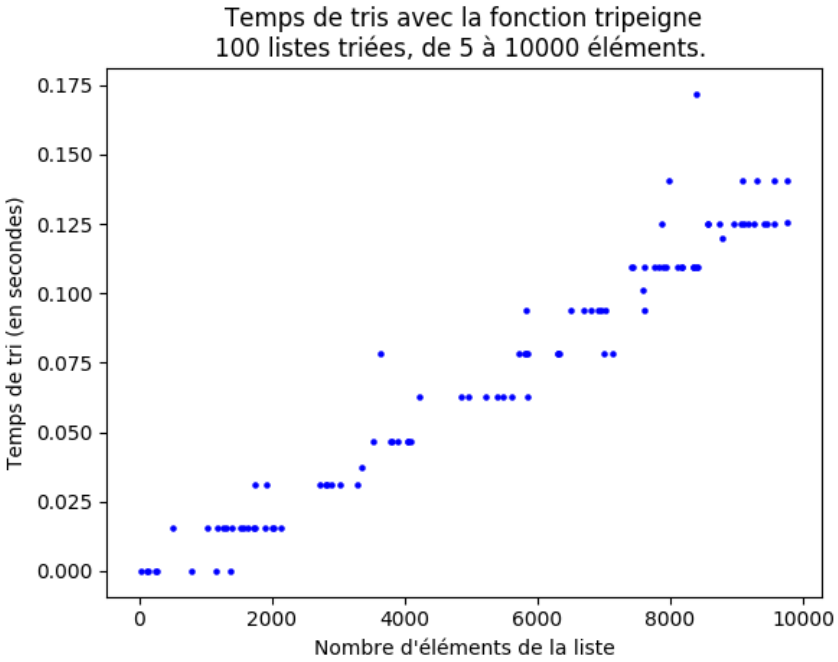
Tri pair/impair (complexité moyenne : $\ominus(N^2)$)

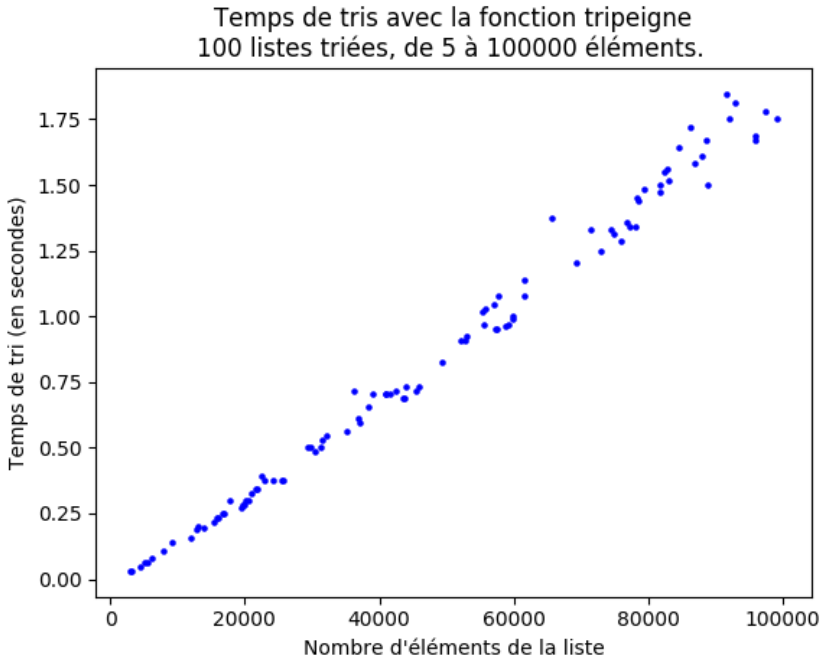
```
def tripairimpair(t):
    """
    algorithme direct de tri pair/impair
    """
    triOK=False
    while not triOK:
        triOK=True
        for i in range(0,len(t)-1,2):  #tri pair-impair
            if t[i+1]<t[i]:
                echange(t,i,i+1)
                triOK=False
        for i in range(1,len(t)-1,2):   #tri impair-pair
            if t[i+1]<t[i]:
                echange(t,i,i+1)
                triOK=False
```



Tri à peigne (complexité moyenne : $\ominus (N\ log(N))$)

```
def tripeigne(t,coeffreduc=1.3):
    """
    algorithme direct de tri à peigne
    """
    largeurpeigne=len(t)
    triOK=True
    while largeurpeigne>1 or not triOK:
        largeurpeigne=max(1,int(largeurpeigne/coeffreduc))
        i=0
        triOK=True
        while i<=len(t)-largeurpeigne-1:
            if t[i]>t[i+largeurpeigne]:
                echange(t,i,i+largeurpeigne)
                triOK=False
            i+=1
```





Tri fusion (complexité moyenne : $\ominus\ (N\ log(N))$)

```
def trifusion(t1,t2):
    """
    tri par fusion de deux listes ordonnées
    (suppression successive des termes utilisés avec syntaxe épurée)
    """
    t=[]
    for k in range(len(t1)+len(t2)):
        t.append(t1.pop(0) if ((t1!=[] and t2!=[] and t1[0]<=t2[0]) or t2==[]) else t2.pop(0))
    return t

def trifusion_init(t):
    """
    algorithme récursif de tri par fusion d'une liste quelconque
    """
    milieu=len(t)//2
    l1=t[0:milieu]
    l2=t[milieu:len(t)]
    return (trifusion(l1,l2) if max(len(l1),len(l2))<=1 else
trifusion(trifusion_init(l1),trifusion_init(l2)))
```