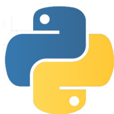
tri par insertion

tri à bulles

tri rapide

tri fusion

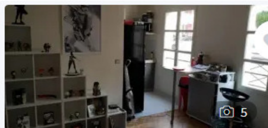
Algorithmes de tris



Problématique : Un étudiant cherche un logement...

Algorithmes de tris

- tri par insertion
- tri à bulles
- tri rapide
- tri fusion

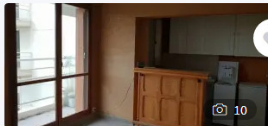


Appartement

1 p | 35,09 m² | 1 asc

760 € cc

St Germain en Laye



Appartement

1 p | 24,5 m² | 1 asc

670 € cc

Saint Germain en Laye

Appartement

1 p | 23,03 m² | 4 etg

740 € cc

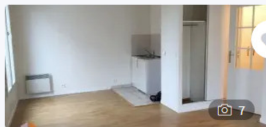
Saint Germain en Laye

Appartement

1 p | 15 m²

590 € cc

St Germain en Laye

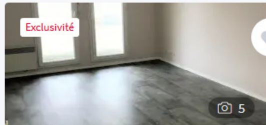


Appartement

1 p | 31,38 m² | 1 asc

705 € cc

Saint-Germain-en-Laye

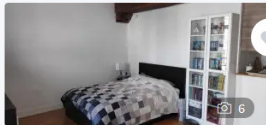


Appartement

1 p | 31,66 m² | 1 asc

665 € cc

Saint-Germain-en-Laye

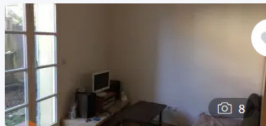


Appartement

1 p | 29,34 m² | 3 etg

790 € cc

Saint Germain en Laye

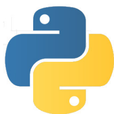


Appartement

1 p | 26,66 m² | 1 asc

733 € cc

Saint-Germain-en-Laye



Algorithmes de tris

tri par insertion
tri à bulles
tri rapide
tri fusion

La recherche lui fournit 181 biens à louer à Saint Germain en Laye. Les résultats tiennent sur plusieurs pages. Pour faciliter la recherche, le site propose néanmoins plusieurs options permettant de trier les différents biens proposés selon plusieurs critères :

- la surface habitable ;
- le prix ;
- le nombre de pièces ;
- la date de mise en ligne...

Pour effectuer ces différents tris le site utilise un algorithme de tris qui a pour objectif principal d'optimiser la durée du tri. (Si c'est trop long les utilisateurs vont se détourner de ce site...)

tri par insertion
tri à bulles
tri rapide
tri fusion

1

tri par insertion

Tri par insertion

Algorithmes de tris

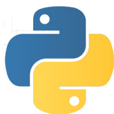
tri par insertion

tri à bulles

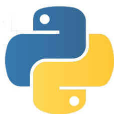
tri rapide

tri fusion

Le tri par insertion est le plus naturel. C'est la méthode utilisée spontanément pour trier des cartes dans sa main ou un tas de copies.
Si les i premières cartes sont triées (par ordre croissant par exemple) on prend la i+1 ^{e}carte et on la fait "descendre" jusqu'à sa place. Ce tris s'effectue en place, c'est-à-dire qu'il ne demande pas d'autre liste numérique que celle que l'on trie. Son coût en mémoire est donc constant.



Le tri par insertion est le plus naturel. C'est la méthode utilisée spontanément pour trier des cartes dans sa main ou un tas de copies.
Si les i premières cartes sont triées (par ordre croissant par exemple) on prend la i+1^{e} carte et on la fait "descendre" jusqu'à sa place. Ce tris s'effectue en place, c'est-à-dire qu'il ne demande pas d'autre liste numérique que celle que l'on trie. Son coût en mémoire est donc constant.



Écriture en Python (tri dans l'ordre croissant)


```
def tri1(t):
    for i in range(1,len(t)):
        j=i
        x=t[i]
        while j>0 and t[j-1]>x:
            t[j]=t[j-1]
            j=j-1
        t[j]=x
    return t
```

Utiliser l'algorithme précédent pour trier la liste :

3 0 1 8 7 2 5 4 9 6

i valeur

1

2

3

4

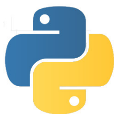
5

6

7

8

9



Complexité du tri par insertion

- Dans le meilleur des cas

Le tableau est alors déjà trié. la boucle `for` parcourt tout le tableau (N-1 passages dans la boucle) et pour chaque passage une seule comparaison à effectuer.

$$\sum_{i=1}^{N-1} 1 = N - 1 \quad \Rightarrow \quad \text{complexité en } O(N)$$

- Dans le pire des cas

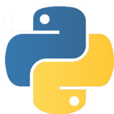
Le tableau est alors trié dans l'ordre décroissant. la boucle `for` parcourt tout le tableau (N-1 passage dans la boucle) et pour chaque indice i du tableau, au plus i comparaisons à effectuer.

$$\sum_{i=0}^{N-1} 1 = \frac{N(N-1)}{2} \quad \Rightarrow \quad \text{complexité en } O(N^2)$$

tri par insertion
tri à bulles
tri rapide
tri fusion

2

tri à bulles



Tri à bulles

Algorithmes de tris

tri par insertion

tri à bulles

tri rapide

tri fusion

```
def triB(t):
    stop=False
    while not stop:
        stop=True
        for i in range(len(t)-1):
            if t[i]>t[i+1]:
                stop=False
                t[i],t[i+1] = t[i+1],t[i]
    return t
```

Utiliser l'algorithme précédent pour trier la liste :

3 0 1 8 7 2 5 4 9 6

passage while

1

2

3

4

5

6

7

8

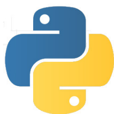
9

Analyser la complexité dans le meilleur et dans le pire des cas de cet algorithme.

Proposer une amélioration de l'algorithme.

3

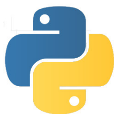
tri rapide



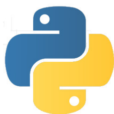
Tri rapide (quicksort)

Il existe de nombreuses formes du quicksort. Une construction « classique » consiste à :

- Définir une fonction **partitionner** qui organise les éléments du tableau autour d'un pivot que l'on peut choisir arbitrairement dans le tableau et renvoie sa position.
- Effectuer récursivement le tri des deux portions de tableau à droite et à gauche du pivot.



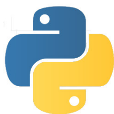

```
def triR(t):

    def partitionner(t,g,d):
        ipivot0=g
        pivot=t[g]
        imem=g
        for i in range(g,d+1):
            if t[i]<pivot:
                imem+=1
                t[i],t[imem]=t[imem],t[i]
        t[ipivot0],t[imem]=t[imem],t[ipivot0]
        return imem
```




```
def quicksort(t,g,d):
    if d>g:
        ipivot=partitionner(t,g,d)
        quicksort(t,g,ipivot)
        quicksort(t,ipivot+1,d)

quicksort(t,0,len(t)-1)
return t
```

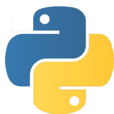
Partitionner la liste :

3 0 1 8 7 2 5 4 9 6



Utiliser l'algorithme précédent pour trier la liste :

3 0 1 8 7 2 5 4 9 6



Autre version du tri rapide

Algorithmes de tris

tri par insertion

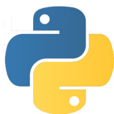
tri à bulles

tri rapide

tri fusion

```
def triR2(t):
    if t==[]:
        return []
    pivot=t.pop()
   p,g,e=[],[],[pivot]
    for x in t:
        if x<pivot:
            p.append(x)
        elif x>pivot:
            g.append(x)
        else:
                        e.append(x)
    return triR2(p) + e + triR2(g)
```

Remarque : Cette version du tri rapide n'est pas "en place" elle demande donc plus d'espace mémoire.



Complexité du tri rapide

Pour un tableau de N valeurs :

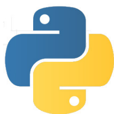
- Complexité dans le pire des cas en $O(N^2)$. Correspond au cas où le pivot choisi vient se positionner systématiquement en bout de tableau.
- Complexité dans le meilleur des cas en $O(N \log N)$. Correspond au cas où le pivot est toujours placé au milieu du tableau.

Remarque : La complexité du tri rapide n'est pas meilleure que celle du tri par insertion. Mais la configuration correspondant au pire des cas n'est pas la même... cela a un impact important sur la performance du tri rapide en pratique.

tri par insertion
tri à bulles
tri rapide
tri fusion

4

tri fusion



Tri fusion

Algorithmes de tris

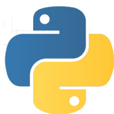
tri par insertion

tri à bulles

tri rapide

tri fusion

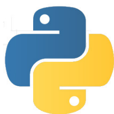
```
def triF(t):

    def fusion(t1,t2):
        if t1==[]:
            return t2
        elif t2==[]:
            return t1
        elif t1[0]<t2[0]:
            return [t1[0]]+fusion(t1[1:],t2)
        else:
            return [t2[0]]+fusion(t1,t2[1:])

    if len(t)<2:
        return t
    m=len(t)//2
    return fusion(triF(t[:m]),triF(t[m:]))
```



Fusionner les deux listes : [3,0,1,8,7] et [2,5,4,9,6]

Utiliser l'algorithme de tri fusion pour trier la liste

3 0 1 8 7 2 5 4 9 6

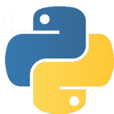


Il s'agit à nouveau d'un tri récursif, mais cette fois-ci, on définit une opération de fusion de 2 listes.

Pour un tableau de taille N, la complexité est en $O(N \log N)$. (dans le meilleurs et le pire des cas)

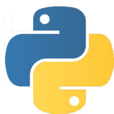
La complexité du tri fusion est optimale : On peut montrer que, pour un tri par comparaison, dans le pire des cas la complexité ne peut être meilleure que $O(N \log N)$.

Le tri fusion n'est pas un tri "en place" il est donc gourmand en espace mémoire.



Comparaison des performances

Remarque : La méthode sort appliquée à une liste permet de trier automatiquement la liste dans l'ordre croissant



Exercice : Détermination de la médiane d'une liste triée

Algorithmes de tris

tri par insertion

tri à bulles

tri rapide

tri fusion

La médiane m d'un ensemble de nombres est une valeur située « au milieu » de l'ensemble :

- Si la liste comporte $2n+1$ éléments ($n \in \mathbb{N}$), la médiane est unique et a le rang $i = n+1$.
- Par contre, si elle comporte $2n$ éléments, il y a une infinité de médianes possibles comprises entre deux valeurs : celle de rang n et celle de rang $n+1$. Dans cet exercice on définit la médiane comme la moyenne arithmétique des deux valeurs extrêmes

1. Définir une fonction **listeTriee** qui prend comme argument une liste de valeurs numériques et retourne `True` si la liste est triée (dans l'ordre croissant ou décroissant) et `False` sinon.
2. Définir une fonction **mediane** qui prend comme argument une liste triée et valeur de la médiane définit ci-dessus.