Algorithme du tri par insertion

Le principe est de considérer qu'une partie située en début de liste est déjà triée (cette partie est initialement vide), ensuite on parcourt le reste de la liste et on insère chaque élément qu'on rencontre dans la partie déjà triée.

on parcourt la liste à partir du premier élément

Algorithme du tri par insertion

Le principe est de considérer qu'une partie située en début de liste est déjà triée (cette partie est initialement vide), ensuite on parcourt le reste de la liste et on insère chaque élément qu'on rencontre dans la partie déjà triée.

- on parcourt la liste à partir du premier élément
- chaque élément rencontré est inséré à la bonne position en début de liste

Algorithme du tri par insertion

Le principe est de considérer qu'une partie située en début de liste est déjà triée (cette partie est initialement vide), ensuite on parcourt le reste de la liste et on insère chaque élément qu'on rencontre dans la partie déjà triée.

- on parcourt la liste à partir du premier élément
- chaque élément rencontré est inséré à la bonne position en début de liste
- Cette insertion peut se faire en échangeant cet élément avec son voisin de gauche tant qu'il lui est supérieur

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par insertion sur cette liste

Début (triée)	Fin (à trier)	

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par insertion sur cette liste

Début (triée)	Fin (à trier)
[	12,10,18,15,14]

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par insertion sur cette liste

Début (triée)	Fin (à trier)
`[`	`12,10,18,15,14]`
`[12,`	`10,18,15,14]`

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par insertion sur cette liste

Début (triée)	Fin (à trier)
`[`	`12,10,18,15,14]`
`[12,`	`10,18,15,14]`
`[12,10,`	`18,15,14]`

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par insertion sur cette liste

Début (triée)	Fin (à trier)	
`[`	`12,10,18,15,14]`	
`[12,`	`10,18,15,14]`	
`[12,10,`	`18,15,14]`	Insertion du 10

C7 Algorithmes de tri

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par insertion sur cette liste

Début (triée)	Fin (à trier)	
`[`	`12,10,18,15,14]`	
`[12,`	`10,18,15,14]`	
`[12,10,`	`18,15,14]`	Insertion du 10
`[10,12`	`18,15,14]`	

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par insertion sur cette liste

Début (triée)	Fin (à trier)	
`[`	`12,10,18,15,14]`	
`[12,`	`10,18,15,14]`	
`[12,10,`	`18,15,14]`	Insertion du 10
`[10,12`	`18,15,14]`	
`[10,12`	`18,15,14]`	

C7 Algorithmes de tri

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par insertion sur cette liste

Début (triée)	Fin (à trier)	
`[`	`12,10,18,15,14]`	
`[12,`	`10,18,15,14]`	
`[12,10,`	`18,15,14]`	Insertion du 10
`[10,12`	`18,15,14]`	
`[10,12`	`18,15,14]`	
`[10,12,18`	`15,14]`	18 déjà placé

C7 Algorithmes de tri

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par insertion sur cette liste

Début (triée)	Fin (à trier)	
`[`	`12,10,18,15,14]`	
`[12,`	`10,18,15,14]`	
`[12,10,`	`18,15,14]`	Insertion du 10
`[10,12`	`18,15,14]`	
`[10,12`	`18,15,14]`	
`[10,12,18`	`15,14]`	18 déjà placé
`[10,12,18,15`	`14]`	Insertion du 15

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par insertion sur cette liste

Début (triée)	Fin (à trier)	
`[`	`12,10,18,15,14]`	
`[12,`	`10,18,15,14]`	
`[12,10,`	`18,15,14]`	Insertion du 10
`[10,12`	`18,15,14]`	
`[10,12`	`18,15,14]`	
`[10,12,18`	`15,14]`	18 déjà placé
`[10,12,18,15`	`14]`	Insertion du 15
`[10,12,15,18`	`14]`	

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par insertion sur cette liste

Début (triée)	Fin (à trier)	
[	12,10,18,15,14]	
[12,	10,18,15,14]	
[12,10,	18,15,14]	Insertion du 10
[10,12	18,15,14]	
[10,12	18,15,14]	
[10,12,18	15,14]	18 déjà placé
[10,12,18,15	14]	Insertion du 15
[10,12,15,18	14]	
[10,12,14,15,18	]	

Implémentation du tri par insertion en Python

En supposant qu'on dispose déjà de la fonction echange qui intervertir les éléments de la liste situés en donnant leurs indices.

```
1     # Tri par insertion
2     def tri_insertion(liste):
3       for ind in range(1,len(liste)-1):
4         pos = ind
5         while pos>=0 and liste[pos+1]<liste[pos]:
6           liste[pos], liste[pos+1] = liste[pos
                +1], liste[pos]
7           pos=pos-1
```

C7 Algorithmes de tri

Notion de complexité d'un algorithme

Lorsqu'on s'intéresse aux performances d'un algorithme, on fait varier le volume de données traité par l'algorithme et on étudie :

C7 Algorithmes de tri

Notion de complexité d'un algorithme

Lorsqu'on s'intéresse aux performances d'un algorithme, on fait varier le volume de données traité par l'algorithme et on étudie :

- l'évolution du nombre d'opération nécessaires au fonctionnement de l'algorithme, c'est ce qu'on appelle la la compléxité en temps de l'algorithme.

C7 Algorithmes de tri

Notion de complexité d'un algorithme

Lorsqu'on s'intéresse aux performances d'un algorithme, on fait varier le volume de données traité par l'algorithme et on étudie :

- l'évolution du nombre d'opération nécessaires au fonctionnement de l'algorithme, c'est ce qu'on appelle la la compléxité en temps de l'algorithme.
- l'évolution de l'espace mémoire nécessaire au fonctionnement de l'algorithme, c'est ce qu'on appelle la la compléxité spatiale de l'algorithme.

C7 Algorithmes de tri

Complexité linéaire

Complexité linéaire

On dira qu'un algorithme a une complexité en temps linéaire lorsque qu'une multiplication de la taille des données par un facteur k se traduit par une augmentation du temps de calcul par un facteur proche de k.

Complexité linéaire

On dira qu'un algorithme a une complexité en temps linéaire lorsque qu'une multiplication de la taille des données par un facteur k se traduit par une augmentation du temps de calcul par un facteur proche de k.

Par exemple si la complexité est linéaire traiter une liste 10 fois plus grande prendra environ 10 fois plus de temps

Complexité linéaire

On dira qu'un algorithme a une complexité en temps linéaire lorsque qu'une multiplication de la taille des données par un facteur k se traduit par une augmentation du temps de calcul par un facteur proche de k.

Par exemple si la complexité est linéaire traiter une liste 10 fois plus grande prendra environ 10 fois plus de temps

Dans ce cas lorsqu'on trace le graphique du temps de calcul en fonction de la taille des données on obtient une droite.

Complexité linéaire

On dira qu'un algorithme a une complexité en temps linéaire lorsque qu'une multiplication de la taille des données par un facteur k se traduit par une augmentation du temps de calcul par un facteur proche de k.

Par exemple si la complexité est linéaire traiter une liste 10 fois plus grande prendra environ 10 fois plus de temps

Dans ce cas lorsqu'on trace le graphique du temps de calcul en fonction de la taille des données on obtient une droite.

Exemple

Un algorithme de parcourt simple d'une liste (par exemple recherche de minimum ou calcul de moyenne) a une complexité linéaire.

C7 Algorithmes de tri

Complexité quadratique

Complexité quadratique

On dira qu'un algorithme a une complexité en temps quadratique lorsque qu'un multiplication de la taille des données par un facteur k se traduit par une augmentation du temps de calcul par un facteur proche de k^2.

Complexité quadratique

On dira qu'un algorithme a une complexité en temps quadratique lorsque qu'un multiplication de la taille des données par un facteur k se traduit par une augmentation du temps de calcul par un facteur proche de k^2.

Par exemple si la complexité est quadratique traiter une liste 10 fois plus grande prendra environ 100 fois plus de temps

Complexité quadratique

On dira qu'un algorithme a une complexité en temps quadratique lorsque qu'un multiplication de la taille des données par un facteur k se traduit par une augmentation du temps de calcul par un facteur proche de k^2.

Par exemple si la complexité est quadratique traiter une liste 10 fois plus grande prendra environ 100 fois plus de temps

Dans ce cas lorsqu'on trace le graphique du temps de calcul en fonction de la taille des données on obtient une parabole.

Complexité tri insertion

Dénombrons le nombre d'opérations dans le pire des cas, pour une liste de taille n.

Complexité tri insertion

Dénombrons le nombre d'opérations dans le pire des cas, pour une liste de taille n.

boucle for : elle s'exécute $n-1$ fois.

Complexité tri insertion

Dénombrons le nombre d'opérations dans le pire des cas, pour une liste de taille n.

boucle for : elle s'exécute $n-1$ fois.

boucle while : dans le pire des cas, elle exécute d'abord 1 opération, puis 2, puis 3... jusqu'à $n-1$. Or

$$1+2+3+\cdots+n-1=\frac{n\times(n-1)}{2}$$

Complexité tri insertion

Dénombrons le nombre d'opérations dans le pire des cas, pour une liste de taille n.

boucle for : elle s'exécute $n-1$ fois.

boucle while : dans le pire des cas, elle exécute d'abord 1 opération, puis 2, puis 3... jusqu'à $n-1$. Or

$$1+2+3+\cdots+n-1=\frac{n\times(n-1)}{2}$$

Le terme de plus haut degré de l'expression $\frac{n\times(n-1)}{2}$ est de degré 2 : le nombre d'opérations effectuées est donc proportionnel au **carré** de la taille des données d'entrée. Ceci démontre que le tri par insertion est de complexité **quadratique** noté $O(n^2)$.

Complexité tri insertion

Dénombrons le nombre d'opérations dans le pire des cas, pour une liste de taille n.

boucle for : elle s'exécute $n-1$ fois.

boucle while : dans le pire des cas, elle exécute d'abord 1 opération, puis 2, puis 3... jusqu'à $n-1$. Or

$$1+2+3+\cdots+n-1=\frac{n\times(n-1)}{2}$$

Le terme de plus haut degré de l'expression $\frac{n\times(n-1)}{2}$ est de degré 2 : le nombre d'opérations effectuées est donc proportionnel au **carré** de la taille des données d'entrée. Ceci démontre que le tri par insertion est de complexité **quadratique** noté $O(n^2)$.

Dans le cas (rare, mais il faut l'envisager) où la liste est déjà triée, on ne rentre jamais dans la boucle while : le nombre d'opérations est dans ce cas égal à $n-1$, ce qui caractérise une complexité linéaire.

Complexité tri insertion

Est-on sûr que notre algorithme va s'arrêter ?
Le programme est constitué d'une boucle while imbriquée dans une boucle for.
Seule la boucle while peut provoquer une non-terminaison de l'algorithme.
Observons donc ses conditions de sortie :

Complexité tri insertion

Est-on sûr que notre algorithme va s'arrêter ?
Le programme est constitué d'une boucle while imbriquée dans une boucle for.
Seule la boucle while peut provoquer une non-terminaison de l'algorithme.
Observons donc ses conditions de sortie :

```
while pos>=0 and liste[pos+1]<liste[pos] :
```

Complexité tri insertion

Est-on sûr que notre algorithme va s'arrêter ?
Le programme est constitué d'une boucle while imbriquée dans une boucle for.
Seule la boucle while peut provoquer une non-terminaison de l'algorithme.
Observons donc ses conditions de sortie :

> while pos>=0 and liste[pos+1]<liste[pos] :
>
> La condition ‘liste[pos+1] < liste[pos]‘ ne peut pas être rendue fausse avec certitude.

Complexité tri insertion

Est-on sûr que notre algorithme va s'arrêter ?
Le programme est constitué d'une boucle while imbriquée dans une boucle for.
Seule la boucle while peut provoquer une non-terminaison de l'algorithme.
Observons donc ses conditions de sortie :

> while pos>=0 and liste[pos+1]<liste[pos] :
>
> La condition 'liste[pos+1] < liste[pos]' ne peut pas être rendue fausse avec certitude.
>
> Par contre, la condition pos >= 0 sera fausse dès que la variable pos deviendra négative.
> Or la ligne pos = pos - 1 nous assure que la variable pos diminuera à chaque tour de boucle.
> La condition pos >= 0 deviendra alors forcément fausse au bout d'un certain temps.

Complexité tri insertion

Est-on sûr que notre algorithme va s'arrêter ?
Le programme est constitué d'une boucle while imbriquée dans une boucle for.
Seule la boucle while peut provoquer une non-terminaison de l'algorithme.
Observons donc ses conditions de sortie :

> while pos>=0 and liste[pos+1]<liste[pos] :
>
> La condition 'liste[pos+1] < liste[pos]' ne peut pas être rendue fausse avec certitude.
>
> Par contre, la condition pos >= 0 sera fausse dès que la variable pos deviendra négative.
> Or la ligne pos = pos - 1 nous assure que la variable pos diminuera à chaque tour de boucle.
> La condition pos >= 0 deviendra alors forcément fausse au bout d'un certain temps.
>
> Nous avons donc prouvé la **terminaison** de l'algorithme.

Vocabulaire

On dit que la valeur pos est un **variant de boucle**. C'est une notion théorique (ici illustrée de manière simple par la valeur pos) qui permet de prouver la bonne sortie d'une boucle et donc la terminaison d'un algorithme.

C7 Algorithmes de tri

Algorithme du tri par sélection

Rechercher le plus petit élément de la liste à partir de l'indice 0

C7 Algorithmes de tri

Algorithme du tri par sélection

Rechercher le plus petit élément de la liste à partir de l'indice 0

Echanger cet élément avec le premier de la liste

C7 Algorithmes de tri

Algorithme du tri par sélection

Rechercher le plus petit élément de la liste à partir de l'indice 0

Echanger cet élément avec le premier de la liste

Rechercher le plus petit élément de la liste à partir de l'indice 1

C7 Algorithmes de tri

Algorithme du tri par sélection

Rechercher le plus petit élément de la liste à partir de l'indice 0

Echanger cet élément avec le premier de la liste

Rechercher le plus petit élément de la liste à partir de l'indice 1

Echanger cet élément avec le second de la liste

C7 Algorithmes de tri

Algorithme du tri par sélection

Rechercher le plus petit élément de la liste à partir de l'indice 0

Echanger cet élément avec le premier de la liste

Rechercher le plus petit élément de la liste à partir de l'indice 1

Echanger cet élément avec le second de la liste

Et ainsi de suite jusqu'à ce que la liste soit entièrement triée

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par sélection sur cette liste

Sélection du plus petit élément depuis l'indice 0 : `[12,10,18,15,14]`

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par sélection sur cette liste

Sélection du plus petit élément depuis l'indice 0 : `[12,10,18,15,14]`

Placement en première position de liste : `[10,12,18,15,14]`

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par sélection sur cette liste

Sélection du plus petit élément depuis l'indice 0 : `[12,10,18,15,14]`

Placement en première position de liste : `[10,12,18,15,14]`

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par sélection sur cette liste

- Sélection du plus petit élément depuis l'indice 0 : `[12,10,18,15,14]`
- Placement en première position de liste : `[10,12,18,15,14]`
- Sélection du plus petit élément depuis l'indice 1 : `[10,12,18,15,14]`

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par sélection sur cette liste

- Sélection du plus petit élément depuis l'indice 0 : `[12,10,18,15,14]`
- Placement en première position de liste : `[10,12,18,15,14]`
- Sélection du plus petit élément depuis l'indice 1 : `[10,12,18,15,14]`
- Placement en deuxième position de liste : `[10,12,18,15,14]`

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par sélection sur cette liste

- Sélection du plus petit élément depuis l'indice 0 : `[12,10,18,15,14]`
- Placement en première position de liste : `[10,12,18,15,14]`
- Sélection du plus petit élément depuis l'indice 1 : `[10,12,18,15,14]`
- Placement en deuxième position de liste : `[10,12,18,15,14]`
- Sélection du plus petit élément depuis l'indice 2 : `[10,12,18,15,14]`

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par sélection sur cette liste

- Sélection du plus petit élément depuis l'indice 0 : `[12,10,18,15,14]`
- Placement en première position de liste : `[10,12,18,15,14]`
- Sélection du plus petit élément depuis l'indice 1 : `[10,12,18,15,14]`
- Placement en deuxième position de liste : `[10,12,18,15,14]`
- Sélection du plus petit élément depuis l'indice 2 : `[10,12,18,15,14]`
- Placement en 3^e de liste : `[10,12,14,15,18]`

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par sélection sur cette liste

- Sélection du plus petit élément depuis l'indice 0 : `[12,10,18,15,14]`
- Placement en première position de liste : `[10,12,18,15,14]`
- Sélection du plus petit élément depuis l'indice 1 : `[10,12,18,15,14]`
- Placement en deuxième position de liste : `[10,12,18,15,14]`
- Sélection du plus petit élément depuis l'indice 2 : `[10,12,18,15,14]`
- Placement en 3^e de liste : `[10,12,14,15,18]`
- Sélection du plus petit élément depuis l'indice 3 : `[10,12,14,15,18]`

Exemple

On considère la liste `[12,10,18,15,14]` décrire les étapes d'un tri par sélection sur cette liste

- Sélection du plus petit élément depuis l'indice 0 : `[12,10,18,15,14]`
- Placement en première position de liste : `[10,12,18,15,14]`
- Sélection du plus petit élément depuis l'indice 1 : `[10,12,18,15,14]`
- Placement en deuxième position de liste : `[10,12,18,15,14]`
- Sélection du plus petit élément depuis l'indice 2 : `[10,12,18,15,14]`
- Placement en 3^{e} de liste : `[10,12,14,15,18]`
- Sélection du plus petit élément depuis l'indice 3 : `[10,12,14,15,18]`
- Placement en 4^{e} position de liste : `[10,12,14,15,18]`

Implémentation du tri par sélection en Python

```
1       def tri_selection(lst) :
2       for k in range(len(lst)-1):
3         indice_min = k
4         for i in range(k+1, len(lst)) :
5           if lst[i] < lst[indice_min]:
6             indice_min = i
7         lst[k], lst[indice_min] = lst[indice_min
            ], lst[k]
```

Complexité tri insertion

Dénombrons le nombre d'opérations dans le pire des cas, pour une liste de taille n.

Complexité tri insertion

Dénombrons le nombre d'opérations dans le pire des cas, pour une liste de taille n.

boucle for : elle s'exécute $n - 1$ fois.

Complexité tri insertion

Dénombrons le nombre d'opérations dans le pire des cas, pour une liste de taille n.

boucle for : elle s'exécute $n-1$ fois.

deuxième boucle for imbriquée : elle exécute d'abord 1 opération, puis 2, puis 3... jusqu'à $n-1$. Or

$$1+2+3+\cdots+n-1=\frac{n\times(n-1)}{2}$$

Complexité tri insertion

Dénombrons le nombre d'opérations dans le pire des cas, pour une liste de taille n.

boucle for : elle s'exécute $n-1$ fois.

deuxième boucle for imbriquée : elle exécute d'abord 1 opération, puis 2, puis 3... jusqu'à $n-1$. Or

$$1+2+3+\cdots+n-1=\frac{n\times(n-1)}{2}$$

Ceci est bien un polynôme du second degré, ce qui confirme que la complexité de ce tri est quadratique.

C7 Algorithmes de tri

A retenir !

Les algorithmes de tri par insertion ou par sélection ont une complexité quadratique.