C9 Algorithmes de tri

Généralités

Les algorithmes de tris sont fondamentaux en informatique, ils prennent en entrée une suite de valeurs (nombres, mots, ...) qu'on peut *comparer* entre eux (relation supérieur ou égal pour les nombres, ordre alphabétique pour les mots, ...) et fournissent en sortie cette liste de valeurs ordonnées.

C9 Algorithmes de tri

Généralités

Les algorithmes de tris sont fondamentaux en informatique, ils prennent en entrée une suite de valeurs (nombres, mots, ...) qu'on peut *comparer* entre eux (relation supérieur ou égal pour les nombres, ordre alphabétique pour les mots, ...) et fournissent en sortie cette liste de valeurs ordonnées.

⚠ Spécificités de Python

- Les listes de Python sont modifiées lorsqu'on les passe en paramètre à une fonction, on dit qu'elles sont mutables.

C9 Algorithmes de tri

Généralités

Les algorithmes de tris sont fondamentaux en informatique, ils prennent en entrée une suite de valeurs (nombres, mots, ...) qu'on peut *comparer* entre eux (relation supérieur ou égal pour les nombres, ordre alphabétique pour les mots, ...) et fournissent en sortie cette liste de valeurs ordonnées.

⚠ Spécificités de Python

- Les listes de Python sont modifiées lorsqu'on les passe en paramètre à une fonction, on dit qu'elles sont mutables.
- Par conséquent, une liste passé en paramètre lors d'un tri est modifiée, et on a donc pas besoin d'utiliser une instruction `return` pour récupérer la liste triée dans le programme principal.

C9 Algorithmes de tri

Généralités

Les algorithmes de tris sont fondamentaux en informatique, ils prennent en entrée une suite de valeurs (nombres, mots, ...) qu'on peut *comparer* entre eux (relation supérieur ou égal pour les nombres, ordre alphabétique pour les mots, ...) et fournissent en sortie cette liste de valeurs ordonnées.

⚠ Spécificités de Python

- Les listes de Python sont modifiées lorsqu'on les passe en paramètre à une fonction, on dit qu'elles sont mutables.
- Par conséquent, une liste passé en paramètre lors d'un tri est modifiée, et on a donc pas besoin d'utiliser une instruction `return` pour récupérer la liste triée dans le programme principal.
- D'autres types de données de Python (par exemple les entiers ou les chaines de caractères) sont non mutables, elles ne sont pas modifiées lorsqu'on les passe en paramètre à une fonction.

C9 Algorithmes de tri

Algorithme du tri par sélection

- Rechercher le plus petit élément de la liste

C9 Algorithmes de tri

Algorithme du tri par sélection

- Rechercher le plus petit élément de la liste
- Echanger cet élément avec le premier de la liste

C9 Algorithmes de tri

Algorithme du tri par sélection

- Rechercher le plus petit élément de la liste
- Echanger cet élément avec le premier de la liste
- Recherche le second plus petit élément de la liste

C9 Algorithmes de tri

Algorithme du tri par sélection

- Rechercher le plus petit élément de la liste
- Echanger cet élément avec le premier de la liste
- Recherche le second plus petit élément de la liste
- Echanger cet élément avec le second de la liste

C9 Algorithmes de tri

Algorithme du tri par sélection

- Rechercher le plus petit élément de la liste
- Echanger cet élément avec le premier de la liste
- Recherche le second plus petit élément de la liste
- 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

Exemple

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

1. Sélection du plus petit élément : `[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

1. Sélection du plus petit élément : `[12,10,18,15,14]`
2. 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

1. Sélection du plus petit élément : `[12,10,18,15,14]`
2. Placement en première position de liste : `[10,12,18,15,14]`
3. Sélection du second plus petit élément : `[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

1. Sélection du plus petit élément : `[12,10,18,15,14]`
2. Placement en première position de liste : `[10,12,18,15,14]`
3. Sélection du second plus petit élément : `[10,12,18,15,14]`
4. 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

1. Sélection du plus petit élément : `[12,10,18,15,14]`
2. Placement en première position de liste : `[10,12,18,15,14]`
3. Sélection du second plus petit élément : `[10,12,18,15,14]`
4. Placement en deuxième position de liste : `[10,12,18,15,14]`
5. Sélection du 3^e plus petit élément : `[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

1. Sélection du plus petit élément : `[12,10,18,15,14]`
2. Placement en première position de liste : `[10,12,18,15,14]`
3. Sélection du second plus petit élément : `[10,12,18,15,14]`
4. Placement en deuxième position de liste : `[10,12,18,15,14]`
5. Sélection du 3^{e} plus petit élément : `[10,12,18,15,14]`
6. 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

1. Sélection du plus petit élément : `[12,10,18,15,14]`
2. Placement en première position de liste : `[10,12,18,15,14]`
3. Sélection du second plus petit élément : `[10,12,18,15,14]`
4. Placement en deuxième position de liste : `[10,12,18,15,14]`
5. Sélection du 3^e plus petit élément : `[10,12,18,15,14]`
6. Placement en 3^e de liste : `[10,12,14,15,18]`
7. Sélection du 4^e plus petit élément : `[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

1. Sélection du plus petit élément : `[12,10,18,15,14]`
2. Placement en première position de liste : `[10,12,18,15,14]`
3. Sélection du second plus petit élément : `[10,12,18,15,14]`
4. Placement en deuxième position de liste : `[10,12,18,15,14]`
5. Sélection du 3^e plus petit élément : `[10,12,18,15,14]`
6. Placement en 3^e de liste : `[10,12,14,15,18]`
7. Sélection du 4^e plus petit élément : `[10,12,14,15,18]`
8. Placement en 4^e position de liste : `[10,12,14,15,18]`

Implémentation en Python

En supposant qu'on dispose déjà de :

Implémentation en Python

En supposant qu'on dispose déjà de :

- la fonction `ind_min` qui renvoie l'indice du plus petit des éléments à partir de l'indice `ind`,

Implémentation en Python

En supposant qu'on dispose déjà de :

- la fonction `ind_min` qui renvoie l'indice du plus petit des éléments à partir de l'indice `ind`,
- la fonction `echange` qui intervertit les éléments de la liste situés aux indices `ind` et `ind_min`.

Implémentation du tri par sélection en Python

En supposant qu'on dispose déjà de :

- la fonction `ind_min` qui renvoie l'indice du plus petit des éléments à partir de l'indice `ind`,
- la fonction `echange` qui intervertir les éléments de la liste situés aux indices `ind` et `ind_min`.

```
1 def tri_selection(liste):
2     longueur = len(liste)
3     for ind in range(longueur):
4         ind_min = min_liste(liste,ind)
5         echange(liste,ind,ind_min)
```

C9 Algorithmes de tri

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

C9 Algorithmes de tri

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 sélection 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 sélection 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 sélection 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 sélection 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 sélection 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

Exemple

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

Exemple

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

Exemple

On considère la liste [12,10,18,15,14] décrire les étapes d'un tri par sélection 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 sélection 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 sélection 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 `insere` qui permet d'insérer un élément au bon emplacement dans une liste *déjà triée*,

```
1# Tri par insertion
2def tri_insertion(liste):
3    for ind in range(len(liste)):
4        # insere permet d'insérer au bon
             emplacement entre les indices 0 et ind
5        insere(liste,liste[ind],ind)
```

C9 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 :

C9 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.

C9 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.

C9 Algorithmes de tri

Complexité linéaire

Complexité linéaire

- On dira qu'un algorithme a une complexité en temps linéaire 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.

Complexité linéaire

- On dira qu'un algorithme a une complexité en temps linéaire 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.
- 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'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.
- 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'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.
- 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.

C9 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 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.

Exemple

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

Exemples

- On suppose qu'on dispose d'un algorithme de complexité linéaire travaillant sur une liste, il traite une liste de 1 000 éléments en 0,015 secondes. Donner une estimation du temps de calcul pour une liste de 250 000 éléments.

Exemples

- On suppose qu'on dispose d'un algorithme de complexité linéaire travaillant sur une liste, il traite une liste de 1 000 éléments en 0,015 secondes. Donner une estimation du temps de calcul pour une liste de 250 000 éléments.
  La taille des données a été multiplié par 250, la complexité étant lineaire le temps de calcul sera aussi approximativement multiplié par 250.

Exemples

- On suppose qu'on dispose d'un algorithme de complexité linéaire travaillant sur une liste, il traite une liste de 1 000 éléments en 0,015 secondes. Donner une estimation du temps de calcul pour une liste de 250 000 éléments.
  La taille des données a été multiplié par 250, la complexité étant lineaire le temps de calcul sera aussi approximativement multiplié par 250.
  $0.015 \times 250 = 3.75$, on peut donc prévoir un temps de calcul d'environ 3,75 secondes

Exemples

- On suppose qu'on dispose d'un algorithme de complexité linéaire travaillant sur une liste, il traite une liste de 1 000 éléments en 0,015 secondes. Donner une estimation du temps de calcul pour une liste de 250 000 éléments.
  La taille des données a été multiplié par 250, la complexité étant lineaire le temps de calcul sera aussi approximativement multiplié par 250.
  $0.015 \times 250 = 3.75$, on peut donc prévoir un temps de calcul d'environ 3,75 secondes
- Même question pour un algorithme de complexité quadratique qui traite une liste de 1 000 éléments en 0,07 secondes.

Exemples

- On suppose qu'on dispose d'un algorithme de complexité linéaire travaillant sur une liste, il traite une liste de 1 000 éléments en 0,015 secondes. Donner une estimation du temps de calcul pour une liste de 250 000 éléments.
  La taille des données a été multiplié par 250, la complexité étant lineaire le temps de calcul sera aussi approximativement multiplié par 250.
  $0.015 \times 250 = 3.75$, on peut donc prévoir un temps de calcul d'environ 3,75 secondes
- Même question pour un algorithme de complexité quadratique qui traite une liste de 1 000 éléments en 0,07 secondes.
  La taille des données a été multiplié par 250, la complexité étant quadratique le temps de calcul sera approximativement multiplié par $250^2 = 62500$

Exemples

- On suppose qu'on dispose d'un algorithme de complexité linéaire travaillant sur une liste, il traite une liste de 1 000 éléments en 0,015 secondes. Donner une estimation du temps de calcul pour une liste de 250 000 éléments.
  La taille des données a été multiplié par 250, la complexité étant lineaire le temps de calcul sera aussi approximativement multiplié par 250.
  $0.015 \times 250 = 3.75$, on peut donc prévoir un temps de calcul d'environ 3,75 secondes
- Même question pour un algorithme de complexité quadratique qui traite une liste de 1 000 éléments en 0,07 secondes.
  La taille des données a été multiplié par 250, la complexité étant quadratique le temps de calcul sera approximativement multiplié par $250^2 = 62500$
  $0.07 \times 62\,500 = 4375$, on peut donc prévoir un temps de calcul d'environ $4\,375$ secondes, c'est à dire près d'une heure et 15 minutes !