Introduction à l'informatique

pour les mathématiques, la physique et les sciences computationnelles

Yann Thorimbert

Partie 2 | Chapitre 1
Algorithmes de tri naïfs

Yann Thorimbert

Chapitres du cours (seconde partie du cours)

0. Introduction à la complexité algorithmique

1. Algorithmes de tri naïfs ←

2. Structures de données : tableaux, listes et dictionnaires

3. Tri fusion et récursivité

4. Algorithmes de recherche au sein d'une séquence

5. Algorithmes sur graphes

Rappel

- Nous avons vu plusieurs structures de données permettant d'ordonner ou d'associer des nombres (ou d'autres quantités).
  - Tableaux
  - Listes chaînées
  - Tables de hachage

- Le but de ce chapitre est de découvrir différentes stratégies pour obtenir, à partir d'une séquence de nombres quelconque, une version triée de cette même séquence.

- Dans ce qui suit, nous utiliserons le terme "séquence" de façon générique pour désigner n'importe quel **type abstrait** permettant de lier un indice à une valeur. Nous ne désignons pas une structure de donnée spécifique.

Avant d'entrer dans le vif du sujet | **Retour sur la complexité**

- Nous allons souvent comparer différents algorithmes entre eux en nous référant à leur complexité en temps (cf. chapitre précédent).

- Afin de discuter plus en profondeur de la notion de complexité, abordons ici un algorithme qui nous permet de trouver la valeur minimale au sein d'une séquence de nombres.

Complexité | **Algo de recherche du minimum**

Complexité | **Algo de recherche du minimum**

- Il faut noter que l'algorithme de recherche du minimum est *O*(n) seulement si l'accès aux éléments `L[i]` est *O*(1).

- Dans le cas contraire, il faut en tenir compte ! Par exemple, si la complexité moyenne d'accès aux éléments `L[i]` est $O(n^3)$, alors l'algorithme dans son ensemble est $O(n^4)$.

- *O*(n) dénote le **comportement asymptotique** de la fonction.

- On peut s'intéresser à d'autres complexités que la complexité en temps, comme la complexité en mémoire. Comme ce n'est pas le cas dans ce cours, nous omettons souvent de préciser qu'il s'agit de la complexité en temps.

Algorigrammes | **Algo de recherche du minimum**

Les algorithmes de tri

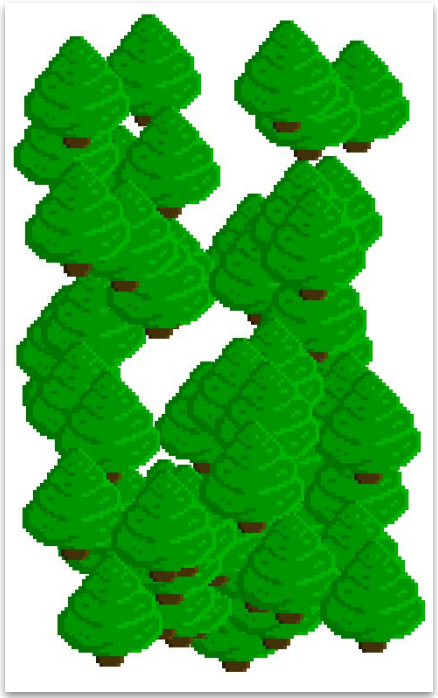
Pourquoi s'intéresser au tri ?

- Comme on l'a vu, les séquences de valeurs sont omniprésentes au sein des programmes informatiques.

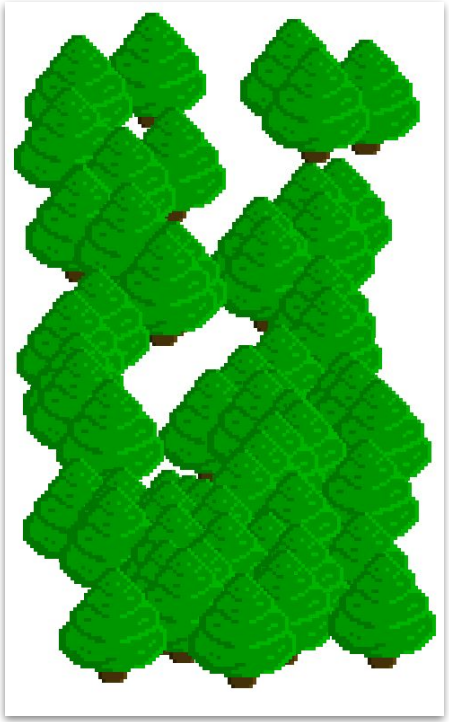
- Dans beaucoup de cas, on désirera chercher des valeurs spécifiques au sein d'une séquence. L'efficacité de telles recherches est impactée par l'ordre des valeurs.

- Dans beaucoup de cas, posséder des données triées offre des garanties désirables sur la structure du problème.

Exemples d'utilisation du tri | **Affichage graphique**



Affichage d'éléments triés en fonction de leur coordonnée verticale.



Exemples d'utilisation du tri | **Calcul de médiane**

- Définition informelle de la médiane d'une séquence de nombres : valeur telle que la moitié des éléments de la séquence sont plus petits, et l'autre moitié plus grands.

- Question : quel est le salaire médian parmi la population suisse ?
  `salaires = [4500, 3850, 7600, 2500, ..., 5650]` # n ≈ 5 millions.

- Pour simplifier l'exemple, supposons un nombre impair de salaires, noté *n*. On suppose également que tous les salaires sont uniques.

Exemples d'utilisation du tri | **Calcul de médiane**

- Quel est le salaire médian parmi la population suisse ?

```
salaires = [4500, 3850, 7600, 2500, ..., 5650] # n ≈ 5 millions.
```

- Une approche : pour chaque élément de la séquence, **compter** le nombre k d'éléments plus petits que lui. Retourner l'élément tel que k = (n -1)/2.

- Une autre approche : ?

Exemples d'utilisation du tri | **Calcul de médiane**

- Quel est le salaire médian parmi la population suisse ?

```
salaires = [4500, 3850, 7600, 2500, ..., 5650] # n ≈ 5 millions.
```

- Une approche : pour chaque élément de la séquence, **compter** le nombre k d'éléments plus petits que lui. Retourner l'élément tel que k = (n -1)/2.

- Une autre approche : **trier la séquence**, puis retourner l'élément d'indice n/2.

Très souvent, un algorithme **commence par effectuer un tri** sur les données en vue d'un traitement subséquent !

Exemples d'utilisation du tri | **Calcul de médiane**

Approche 1: pour chaque élément de la séquence, **compter** le nombre k d'éléments plus petits que lui. Retourner l'élément tel que k = (n -1)/2.

VS

Approche 2 : **trier la séquence**, puis retourner l'élément d'indice n / 2.

Exemples d'utilisation du tri | **Calcul de médiane**

Approche 1: pour chaque élément de la séquence, **compter** le nombre k d'éléments plus petits que lui. Retourner l'élément tel que k = (n -1)/2.

> Pour comparer les deux approches, nous devrons étudier les algorithmes de tri. Mais commençons par l'approche 1.

Approche 2 : **trier la séquence**, puis retourner l'élément d'indice n / 2.

Exemples d'utilisation du tri | **Calcul de médiane**

1ère approche : pour chaque élément de la séquence, compter le nombre k d'éléments plus petits que lui. Retourner l'élément tel que k = (n -1)/2.
Quelle est la complexité de cette approche ?

```
salaires = [4500, 3850, 7600, 2500, … , 5650]
```

```
k = 0
4500 > 3850 ? Oui, k = k + 1.
4500 > 7600 ? Non.
4500 > 3850 ? Oui, k = k + 1.
...
4500 > 5650 ? Non.
```

Exemples d'utilisation du tri | **Calcul de médiane**

1ère approche : pour chaque élément de la séquence, compter le nombre k d'éléments plus petits que lui. Retourner l'élément tel que k = (n -1)/2.
Quelle est la complexité de cette approche ?

```
salaires = [4500, 3850, 7600, 2500, … , 5650]
```

```
k = 0
5650 > 3850 ? Oui, k = k + 1.
5650 > 3850 ? Oui, k = k + 1.
5650 > 7600 ? Non.
5650 > 3850 ? Oui, k = k + 1.
...
```

Exemples d'utilisation du tri | **Calcul de médiane**

1ère approche : pour chaque élément de la séquence, compter le nombre k d'éléments plus petits que lui. Retourner l'élément tel que k = (n -1)/2.
Quelle est la complexité de cette approche ?

```
def mediane_approche1(salaires):
    ?
```

Exemples d'utilisation du tri | **Calcul de médiane**

1ère approche : pour chaque élément de la séquence, compter le nombre k d'éléments plus petits que lui. Retourner l'élément tel que k = (n - 1)/2.
Quelle est la complexité de cette approche ?

Nombre de salaires à examiner ~ n.
Pour chacun, nombre de salaires avec lesquels le comparer ~ n.
⇒ Cet algorithme nécessite un nombre de comparaisons proportionnel à n^2.

Cet algorithme possède une complexité $O(n^2)$.

Exemples d'utilisation du tri | **Calcul de médiane**

- Cet algorithme possède une complexité temporelle de $O(n^2)$.
  Cela signifie que le nombre d'étapes de l'algorithme pour trouver la médiane a un **comportement quadratique** par rapport au nombre de salaires.

- Par rapport à un algorithme en $O(n^3)$ par exemple, il existera toujours une valeur de n pour laquelle cet algorithme est plus performant.

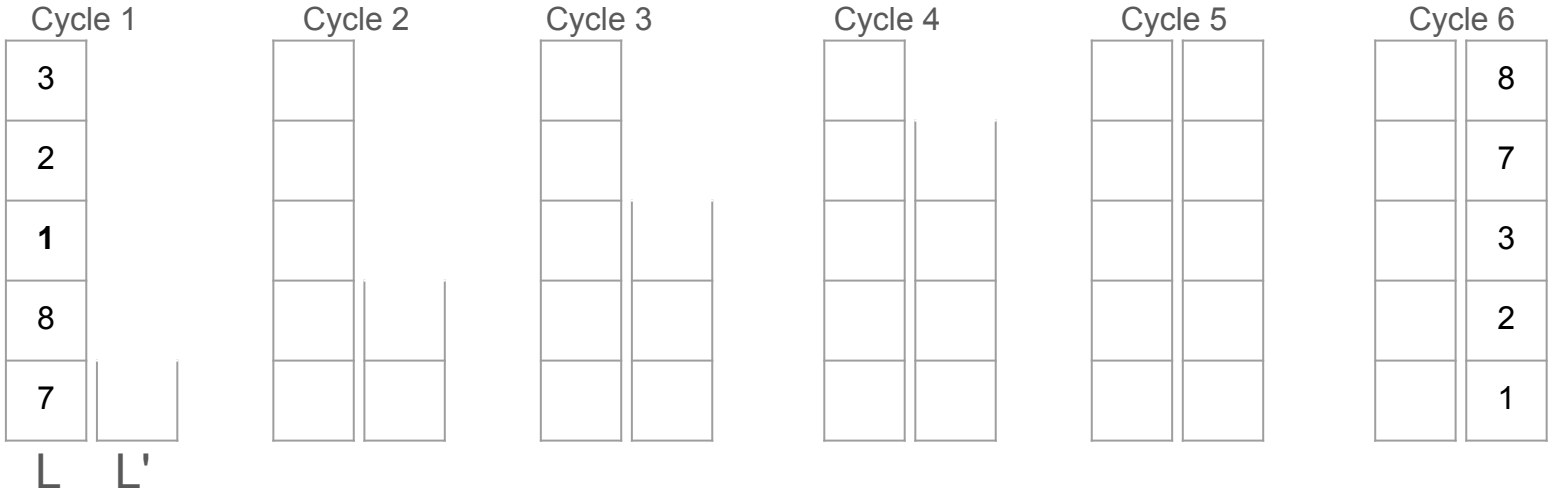
- Quid de l'approche n°2 ?
  ⇒ Cela va dépendre de l'algorithme de tri utilisé. Est-il possible de faire mieux que $O(n^2)$?

Exemple de tri naïf

1. Préparer une séquence vide S' = [ ]
2. Tant que length(S') ≠ n :
   a. m = min(S)
   b. Enlever m de la séquence S
   c. Ajouter m dans la séquence S'

Exemple de tri naïf

1. Préparer une séquence vide S' = [ ]
2. Tant que length(S') ≠ n :
   a. m = min(S)
   b. Enlever m de la séquence S
   c. Ajouter m dans la séquence S'



Exemple de tri naïf

1. Préparer une séquence vide S' = [ ]
2. Tant que length(S') ≠ n :
   a. m = min(S)
   b. Enlever m de la séquence S
   c. Ajouter m dans la séquence S'

Cet algorithme a l'avantage d'être facile à imaginer, mais il serait intéressant de ne pas construire un tableau L ' au fur et à mesure.

Formulons une version dite "**en place**" (*in place*), où le tableau S est **modifié** de telle sorte qu'il soit trié à la fin de l'algorithme.

Tri sélection

7	8	1	3	2

Reprend l'idée de la recherche des minimums successifs.

Cf. slides dévolus à l'**illustration pas à pas**.

Tri sélection | **Pseudocode**

Reprend l'idée de la recherche des minimums successifs :

1. Pour chaque entier i_debut de 0 jusqu'à n-2 :
    a. i_min = indice du minimum entre L[i_debut+1] et L[n-1]
    b. Intervertir les éléments d'indice i_min et i_debut.

Tri sélection | **Pseudocode**

Reprend l'idée de la recherche des minimums successifs :

```
1. Pour chaque entier i_debut de 0 jusqu'à n-2 :
    a. i_min = indice du minimum entre L[i_debut+1] et L[n-1]
    b. Intervertir les éléments d'indice i_min et i_debut.
```

Notez que cette formulation est d'un haut degré d'abstraction.

Cependant, nous avons décrit précédemment l'algorithme de recherche du min.

Tri sélection | **Pseudocode**

Reprend l'idée de la recherche des minimums successifs :

```
1. Pour chaque entier i_debut de 0 jusqu'à n-2 :
    a. i_min = indice du minimum entre L[i_debut+1] et L[n-1]
    b. Intervertir les éléments d'indice i_min et i_debut.
```

Vocabulaire : **swap**(i_min, i_debut)

Tri sélection | **Swap**

- Le tri sélection est un exemple de tri **in place**, car aucune copie de la séquence n'est construite : la séquence elle-même est modifiée.

- Lorsqu'on intervertit deux valeurs, on passe néanmoins par une copie temporaire. Exemple de fonction `swap(i, j, a)` en Python :

```
def swap(i, j, a):
    tmp = a[i] # valeur temporaire pour ne pas "perdre" a[i]
    a[i] = a[j]
    a[j] = tmp
```

Tri sélection | **Complexité en temps**

- Le tri sélection est $O(n^2)$.
- Avec ce tri, nos deux approches pour le calcul de la médiane sont équivalentes du point de vue de la complexité.
- En pratique, le tri sélection est peu utilisé, son intérêt étant principalement pédagogique.

Tri à bulles | **Idée**

- Dans le tri sélection, on fait n - 1 passages sur la séquence (nommés "cycles" dans les exemples détaillés) …
- … et à chacun de ces passages, on effectue un "sous-passage" pour trouver le minimum du reste de la séquence.
- Le tri à bulle exploite le fait que, durant les sous-passages, on peut faire des aménagements pratiques de la séquence.
- À chaque cycle, on compare les éléments voisins tout en "remontant" la séquence, et on les intervertit s'ils sont désordonnés.

Tri à bulles

2
3
1
8
7

Avant d'aller plus loin

- Nous allons d'abord nous assurer que nous comprenons les structures de données permettant d'implémenter des algorithmes.
- En particulier, nous allons examiner les complexités d'accès, d'insertion et de suppression d'éléments au sein des structures les plus courantes.
- Nous reviendrons sur le sujet des algorithmes de tri ensuite.