Chapitre 11 : Algorithmes de tri

Depuis le début de l'informatique et de la programmation, de nombreux algorithmes de tri ont été élaborés. Ils peuvent être regroupés en plusieurs catégories :

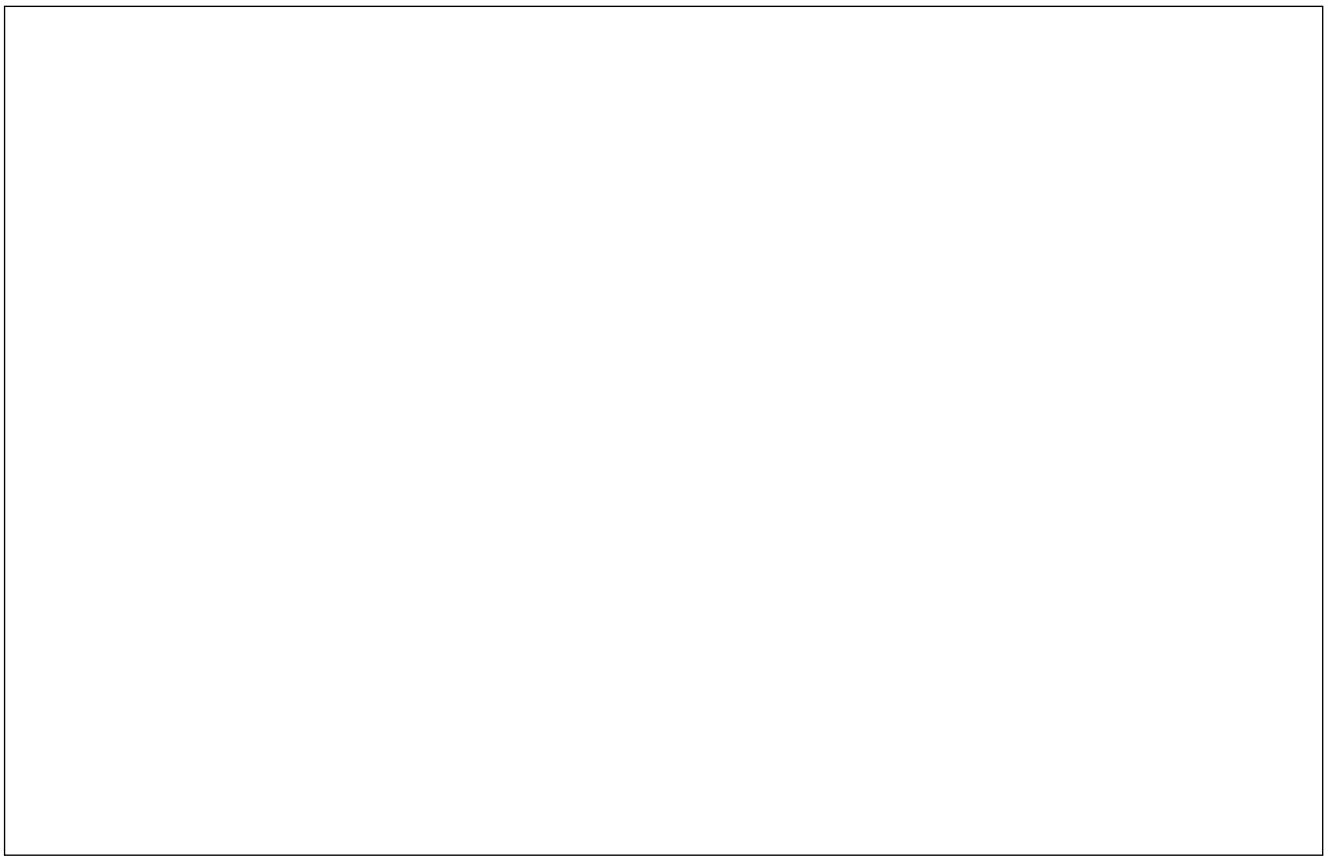
- Les algorithmes de tri quadratiques tels que le tri par sélection ou par insertion
- Les algorithmes de tri utilisant une stratégie « diviser pour régner » tel que le tri fusion
- Les algorithmes de tri sans comparaison tel que le tri par comptage

Dans la suite de ce chapitre, nous supposerons que nous disposons d'une liste `liste` contenant `n` éléments que l'on cherche à trier dans l'ordre croissant.

1. Algorithmes de tri quadratiques

1.1.Tri par sélection

- Principe : on cherche la plus petite donnée de la liste et on la place en première position, puis on cherche le plus petit élément parmi les valeurs restantes et on le place en deuxième, et ainsi de suite.
- Méthode : à l'aide d'une boucle non-conditionnelle d'indice `i`, on cherche pour chaque valeur de `i` le minimum de la sous-liste `liste[i:n]` et on fait l'échange avec `liste[i]` afin de le placer en première position de la sous-liste.
- Algorithme : écrire une fonction `tri_selection` prenant en argument une liste `liste` et qui en effectue le tri par ordre croissant (remarque : la fonction ne retourne pas de résultat puisque la liste à trier est directement modifiée par la fonction)



- Coût : pour chaque valeur de `i`, la recherche du minimum dans la sous-liste `liste[i:n]` entraîne $n-1-i$ comparaisons. Au total, le nombre de comparaisons effectuées lors du processus de tri par sélection est donc :

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

On a donc bien un coût quadratique puisque le nombre d'opérations varie en n^2.

1.2.Tri par insertion

- Principe : en supposant que les k premiers éléments sont déjà triés, on considère l'élément suivant (soit le $k+1$ ème) et on vient l'insérer à la bonne place parmi les k données qui le précèdent.
- Méthode : à l'aide d'une boucle non-conditionnelle d'indice `i`, pour chaque valeur de `i`, on cherche dans la sous-liste `liste[0:i+1]` où on doit placer le terme `liste[i+1]` qu'on appelle la clé. En commençant par la donnée d'indice `i`, on remonte jusqu'à trouver la bonne place (boucle conditionnelle). Toute valeur plus grande que la clé sera ainsi décalée d'un cran vers la droite après chaque comparaison.
- Algorithme : écrire une fonction `tri_insertion` prenant en argument une liste `liste` et qui en effectue le tri par ordre croissant (remarque : la fonction ne retourne pas de résultat puisque la liste à trier est directement modifiée par la fonction)

- Coût : pour une valeur de `i`, dans le pire des cas, l'élément `liste[i+1]` est inférieur à tous ceux de la sous-liste `liste[0:i+1]` ce qui entraîne $i+1$ comparaisons. Le pire des cas correspond ainsi à une liste déjà triée mais dans le sens décroissant. Ainsi, au pire, le nombre total de comparaisons effectuées lors du processus de tri par insertion est donc :

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

On a donc bien un coût quadratique puisque le nombre d'opérations varie en n^2.

1.3.Comparaison des deux types de tri

Ces deux types de tri sont dits « en place » c'est-à-dire qu'aucune nouvelle liste n'est créée. Aucune des deux fonctions écrites ne renvoie un résultat en sortie ; la liste est modifiée en place.

Le tri par insertion est dit « stable » c'est-à-dire que deux éléments de même valeur placés dans un certain ordre avant le tri restent dans le même ordre après le tri. En revanche, ce n'est pas le cas du tri par sélection. Pour le constater, on peut s'appuyer sur l'exemple de la liste `[4,4,3,2,6,5]`.

- Avec le tri par insertion
  - Pour `i=0` avec la clé `4` : `[4,4,3,2,6,5]`
  - Pour `i=1` avec la clé `3` : `[3,4,4,2,6,5]`
  - Pour `i=2` avec la clé `2` : `[2,3,4,4,6,5]`
  - Pour `i=3` avec la clé `6` : `[2,3,4,4,6,5]`
  - Pour `i=4` avec la clé `5` : `[2,3,4,4,5,6]`
- Avec le tri par sélection
  - Pour `i=0` : `[2,4,3,4,6,5]` après échange de `2` et `4`
  - Pour `i=1` : `[2,3,4,4,6,5]` après échange de `3` et `4`
  - Pour `i=2` : `[2,3,4,4,6,5]` après aucun échange
  - Pour `i=3` : `[2,3,4,4,6,5]` après aucun échange
  - Pour `i=4` : `[2,3,4,4,5,6]` après échange de `5` et `6`

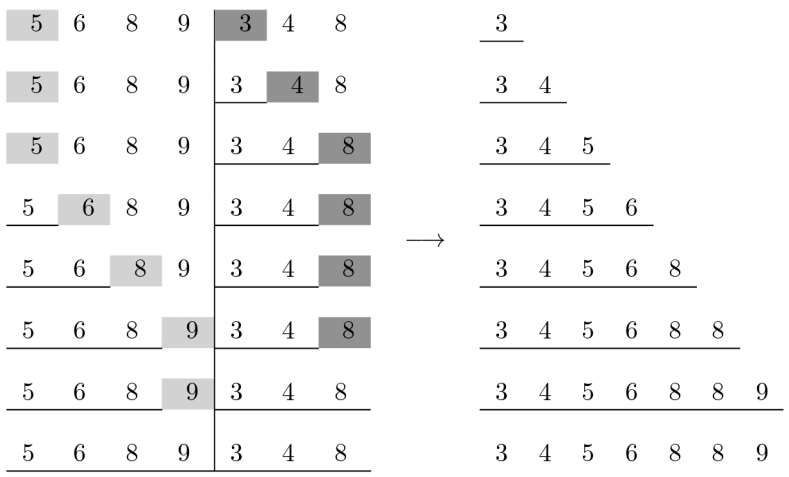
On peut remarquer que, dans tous les cas, pour le tri par sélection d'une liste contenant n éléments, on a un coût quadratique (le nombre de comparaisons est de l'ordre de n^2). Dans le cas du tri par insertion, dans le pire des cas (liste déjà triée à l'envers), on a également un coût quadratique. Toutefois, dans le meilleur des cas c'est-à-dire si la liste est déjà triée (dans le bon sens cette fois), on une comparaison par itération de la boucle conditionnelle donc un coût linéaire (le nombre de comparaisons est de l'ordre de n). L'algorithme de tri par insertion est donc plus efficace que celui par sélection pour une liste « presque triée ».

2. Stratégie « diviser pour régner »

Le principe d'une stratégie « diviser pour régner » est le suivant : on « divise » en réduisant le problème en sous problème du même-type, puis on « règne » en résolvant ces sous-problèmes. Il reste ensuite à combiner les solutions de ces sous-problèmes pour obtenir une solution au problème initial. Un point très important est de ne résoudre qu'une seule fois chaque sous-problème.

Nous allons nous intéresser ici au tri fusion.

- Principe : le tri fusion aurait pu également s'appeler tri dichotomique. En effet, le principe est de partager la liste à trier en deux parties égales à une unité près. On effectue alors un appel récursif sur chacune des deux parties. Une fois triées, les deux listes sont fusionnées afin de former la liste triée attendue.
- Méthode : si partager une liste en deux parties est un problème simple, fusionner deux listes déjà triées pour ne plus en former qu'une seule, également triée (il ne s'agit pas d'une simple concaténation) est en revanche plus complexe. Pour cela, on fusionne deux par deux des parties contiguës de la liste qui ont été triées. On peut voir ci-après un exemple s'appuyant sur deux listes `[5,6,8,9]` et `[3,4,8]`. Les éléments à comparer de chaque partie sont respectivement en gris clair et gris foncé, et on choisit à chaque étape le plus petit. Les éléments choisis sont soulignés.



Une nouvelle liste est ainsi crée.

- Algorithme : écrire une fonction `fusion` prenant en argument deux listes triées `liste1` et `liste2` qui retourne une nouvelle liste résultat de la fusion des deux listes.

Écrire une fonction récursive `tri_fusion` prenant en paramètre une liste `liste` et qui en effectue le tri par tri fusion (la fonction renvoie une liste triée et la liste initiale reste inchangée).

3. Tri sans comparaison

Au lieu d'utiliser des comparaisons, certains algorithmes utilisent la structure des données. C'est le cas des tris par comptage, par base, par paquets...
Nous disposons d'une liste `liste` contenant `n` éléments tous des entiers naturels inférieurs ou égaux à un entier naturel `m` qui est de l'ordre `n`.

- Principe : pour chaque entier naturel inférieur à `m`, on dénombre le nombre d'occurrences dans la liste à trier. On reconstitue une liste triée en tenant compte des résultats précédents.
- Méthode : on crée une liste `compteurs` de `m+1` éléments tous égaux à `0` , le rang d'un élément dans la liste correspondant à un entier naturel possiblement dans la liste `liste` à trier. Si un entier est présent dans `liste`, on vient remplacer par le nombre d'occurences. On recrée la liste triée en ajoutant au fur et à mesure chaque entier naturel autant de fois qu'indiqué dans la liste `compteurs`.
- Algorithme : écrire une fonction `comptage` prenant en argument une liste `entiers` composée d'entiers naturels ainsi qu'un entier naturel `m` et renvoyant une liste de longueur `m+1` telle que pour tout entier `k` compris entre `0` et `m`, l'élément d'indice `k` a pour valeur le nombre d'occurences de l'entier `k` dans la liste `entiers`.

Écrire une fonction `tri`, d'arguments une liste d'entiers naturels `liste` à trier ainsi qu'un entier `m` et renvoyant la liste triée dans l'ordre croissant.