

Les données en informatique sont comme les contenus de vos tiroirs : sans être maniaque, si on range un peu, on retrouve plus facilement ce qu'on cherche. Voici trois algorithmes simples et très classiques de tris de tableaux à une dimension.

Pour chacun de ces algorithmes, on suppose qu'on prend en données un nombre entier n , puis n nombres entiers que l'on place dans un tableau t , et que l'on veut faire un tri par ordre croissant. Les indices sont supposés commencer à zéro pour faciliter la traduction en C.

1 Structure générale du programme

On veut pouvoir saisir les valeurs à rentrer dans un tableau, les afficher, trier le tableau, afficher le résultat. Éventuellement, on veut pouvoir recommencer plusieurs fois ces opérations.

On va décomposer le programme en plusieurs fonctions : une pour la saisie, une pour l'affichage, une pour chaque tri, et une fonction principale (`main`) pour gérer le tout.

Écrivez les fonctions de saisie et d'affichage, et un premier embryon de `main`.

2 Tri par sélection-échange

Idée informelle : placer le plus petit élément en position 0 ; placer le plus petit de ceux qui restent en position 1 ; etc.

Attention : le tableau est trié "sur place", c'est-à-dire qu'on n'utilise pas de tableau annexe pour trier. On doit donc faire un échange de valeurs pour placer un élément là où on veut.

- chercher le numéro de case du plus petit élément du tableau
- si ce numéro n'est pas 0, échanger les valeurs de cet élément et de $t[0]$
- chercher le numéro de case du plus petit élément parmi $t[1], t[2], \dots, t[n-1]$
- si ce numéro n'est pas 1, échanger les valeurs de cet élément et de $t[1]$
- etc ...
- chercher le numéro de case du plus petit élément parmi $t[i], t[i+1], \dots, t[n-1]$
- si ce numéro n'est pas i , échanger les valeurs de cet élément et de $t[i]$
- etc ...
- chercher le numéro de case du plus petit élément parmi $t[n-2], t[n-1]$
- si ce numéro n'est pas $n-2$, échanger les valeurs de cet élément et de $t[n-1]$

Écrivez la fonction qui réalise ce tri.

3 Tri par bulles

Idée informelle : pousser le plus grand élément au bout du tableau par des échanges successifs de proche en proche ; pousser le plus grand de ceux qui restent de la même façon ; etc.

Le tableau est trié "sur place", c'est-à-dire qu'on n'utilise pas de tableau annexe pour trier.

3.1 version simple

- comparer $t[0]$ et $t[1]$; s'ils sont mal rangés, les échanger.
- comparer $t[1]$ et $t[2]$; s'ils sont mal rangés, les échanger.
- ...
- comparer $t[n-2]$ et $t[n-1]$; s'ils sont mal rangés, les échanger.
- On remarque qu'à ce moment-là, le plus grand élément est dans $t[n-1]$.
- comparer $t[0]$ et $t[1]$; s'ils sont mal rangés, les échanger.
- comparer $t[1]$ et $t[2]$; s'ils sont mal rangés, les échanger.
- ...
- comparer $t[n-3]$ et $t[n-2]$; s'ils sont mal rangés, les échanger.
- On remarque que ... ?

- comparer $t[0]$ et $t[1]$; s'ils sont mal rangés, les échanger.
comparer $t[1]$ et $t[2]$; s'ils sont mal rangés, les échanger.
...
- comparer $t[i-1]$ et $t[i]$; s'ils sont mal rangés, les échanger.
On remarque que ...?
- etc ...
- comparer $t[0]$ et $t[1]$; s'ils sont mal rangés, les échanger.
On remarque que ...?

Écrivez la fonction qui réalise ce tri.

3.2 version optimisée

Si aucun échange n'a eu lieu au cours d'un parcours, c'est que le tableau est déjà entièrement rangé : les étapes restantes sont donc inutiles. Modifiez l'algorithme précédent pour qu'il fasse cette optimisation.

4 Tri par comptage

On veut conserver le tableau d'origine inchangé, et avoir à côté une version triée. La méthode consiste à utiliser trois tableaux :

- le tableau t d'origine;
- un tableau c d'entiers dans lequel on compte le nombre de prédécesseurs de chaque élément de t ;
- un tableau $tsor$ dans lequel on range les éléments dans l'ordre.

Écrivez une fonction qui exécute ce tri, sachant que l'algorithme se compose de deux phases successives :

- comptage des prédécesseurs :
Le tableau c est initialisé à 0 au départ. Lorsqu'on compare deux éléments $t[i]$ et $t[j]$, on ajoute 1 à la case de c qui correspond au plus grand des deux. Ainsi, à la fin, la case $c[i]$ contient le nombre d'éléments qui sont plus petits que $t[i]$.
A vous de trouver un algorithme qui ne fait pas de comparaisons inutiles.
- remplissage du tableau final :
Connaissant le nombre de prédécesseurs de $t[i]$, il est facile d'en déduire à quelle place on doit copier sa valeur dans $tsor$. A vous de trouver comment faire ça en une seule boucle de 0 à $n-1$ (qui ne contient pas d'autre boucle).