Algorithmes de tri

Nicolas Gast

20 mars 2007

Exercice 1. Tri par sélection

Le tri par sélection est une des méthodes de tri les plus simples. On commence par rechercher le plus grand élément du tableau que l'on va mettre à sa place : c'est à dire l'échanger avec le dernier élément. Puis on recommence avec le deuxième élément le plus grand, etc...

a. Écrivez une procédure `maxi` qui prend en entrée un tableau `t` et un nombre `n` et qui vous rend le maximum et l'indice du maximum parmi ses `n` premiers éléments du tableau.

Exemple :

```
maxi([1,2,32,1,2,4,60], 5);
                                    32,3
maxi([1,2,32,1,2,4,60], 7);
                                    60,7
```

b. En s'inspirant de la fonction `maxi`, écrivez une fonction `tri_selec` qui tri un tableau selon la méthode de tri par sélection.

c. Pour un tableau de taille n, combien de comparaisons fait votre algorithme ? Combien d'échanges fait votre algorithme (dans le meilleur et le pire cas) ?

Exercice 2. Tri par insertion

Le tri par insertion est généralement le tri que l'on utilise pour classer des documents : on commence par prendre le premier élément à trier que l'on place en position 1. Puis on insère les éléments dans l'ordre en plaçant chaque nouvel élément à sa bonne place.

Pour procéder à un tri par insertion, il suffit de parcourir une liste : on prend les éléments dans l'ordre. Ensuite, on les compare avec les éléments précédents jusqu'à trouver la place de l'élément qu'on considère. Il ne reste plus qu'à décaler les éléments du tableau pour insérer l'élément considéré à sa place dans la partie déjà triée.

Par exemple, si on veut trier le tableau `[10,1,5,19,3,3]`, on obtient successivement :

10, 1,5,19,3,3
1,10, 5,19,3,3
1,5,10, 19,3,3
1,5,10,**19**, 3,3
1,**3**,5,10,19, 3
1,3,**3**,5,10,19

a. Écrire une procédure `tri_insert` qui trie un tableau selon la méthode du tri par insertion.

b. Quel est le nombre de comparaisons effectuées par votre tri ? Quel est le nombre d'échange (pire cas, meilleur cas, moyenne) ?

Exercice 3. Tests

Pour mesurer le temps d'exécution d'un programme, on peut utiliser la commande `time` :

```
> st := time():
> for i from 1 to 100000 do j := 1; od:
> time()-st;
                                    0.032
```

Pour tirer un tableau aléatoire de n éléments, on peut utiliser la commande `[seq(rand(), i=1..n)]`.

a. En utilisant la commande `time`, mesurer le temps pris par les différents algorithmes de tri.

b. Écrire un programme `temps := proc(f,n,p)` qui mesure le temps d'exécution de la fonction `f` appliquée `p` fois à un tableau aléatoire `[seq(rand(),i=1..n)]` et qui rend le temps moyen que prend `f` sur un tableau de taille `n`.

Appliquer cette fonction à vos algorithmes de tri.

Exercice 4. Tri rapide

Le tri rapide est une des méthodes de tri les plus rapide. L'idée est de prendre un élément au hasard (par exemple le premier) que l'on appelle pivot et de le mettre à sa place définitive en plaçant tous les éléments qui sont plus petits à sa gauche et tous ceux qui sont plus grands à sa droite.

On recommence ensuite le tri sur les deux sous-tableaux obtenus jusqu'à ce que le tableau soit trié.

Exemple :

`[10,1,5,19,3,3,2,17]`	le pivot choisit est 10.
`[1,5,3,3,2,` **`10`**`, 19,17]`	On place les éléments plus petits que 10 à gauche et les plus grands à droite
`[1,5,3,3,2]` et `[19,17]`	il reste deux sous-tableaux à trier

a. (sur le papier) écrire un algorithme récursif `tri_rapide` triant un tableau `t` en utilisant l'algorithme.

b. Combien de comparaisons fait votre algorithme (pire cas, meilleur cas, moyenne) ?

c. Implémenter cette fonction en maple.

d. Mesurer les performances du tri rapide sur des listes de taille 5 ,10, 100, 1000 et 10000 et comparer les résultat aux autres algorithmes.

Proposer une méthode pour améliorer le temps pris par le tri rapide.