

# TD de Programmation - Corrigé de l'interrogation 2

L2.1 MathInfo

25 Novembre 2009

## Exercice 1 - Question de cours

Citez trois algorithmes classiques de tri que vous connaissez. Pour chacun donner leur complexité en temps.

Voici quelques exemples de tri classiques:

Tri	Complexité en temps
Tri à bulles	$\mathcal{O}(n^2)$
Tri par insertion	$\mathcal{O}(n^2)$
Tri fusion	$\mathcal{O}(n \log(n))$
Tri par sélection	$\mathcal{O}(n^2)$

## Exercice 2 - Recherche du $k^{\text{ième}}$ élément

Dans le tableau  $[4, 2, 8, 12, 3, 7]$ , le plus petit élément est 2, le deuxième plus petit élément est 3, le cinquième plus petit élément est 8, etc... mais il n'y a ni septième, ni huitième, ni ... plus petit élément puisque le tableau est de taille 6.

1. Ecrire une fonction `int plus_petit(int t[], int taille)` qui renvoie d'indice du plus petit élément d'un tableau d'entiers. Quelle est sa complexité en temps et en espace ?
2. Ecrire une fonction `int plus_petit_2(int t[], int taille)` qui renvoie d'indice du deuxième plus petit élément d'un tableau d'entiers (cette fonction renvoie -1 si le deuxième plus petit élément n'existe pas). Quelle est sa complexité en temps et en espace ?
3. Ecrire une fonction `int plus_petit_k(int t[], int taille, int k)` qui renvoie d'indice du  $k^{\text{ième}}$  plus petit élément d'un tableau d'entiers (cette fonction renvoie -1 si le  $k^{\text{ième}}$  plus petit élément n'existe pas). Quelle est sa complexité en temps et en espace ?

## Exercice 3 - Tri de Shell

1. Décrire brièvement le fonctionnement du tri de Shell. En quoi est-ce une amélioration du tri par insertion?  
*On sait que le tri est insertion est efficace lorsque le tableau de départ est déjà presque trié, mais il est en moyenne peu efficace car il n'échange que des éléments consécutifs dans le tableau. Le tri de Shell est une amélioration du tri par insertion. Le principe est en le suivant: on applique le tri par insertion à des sous-tableaux dont les éléments sont à distance  $h_n$  les uns des autres. L'entier  $h_n$  est appelé la période et diminue au cours de l'algorithme jusqu'à valoir 1. Lorsque la période 1 est atteinte, on effectue un tri par insertion classique et les traitements précédents font qu'à cet instant, le tableau est presque trié (d'où l'efficacité du tri par insertion).*
2. Dérouler l'algorithme du tri de Shell sur l'exemple suivant. Vous préciserez la suite des périodes  $h$  utilisées et mettez en évidence les sous-tableaux considérés à chacune des étapes du tri.  
*On utilise la suite des périodes  $2^n$ . Comme le tableau est de taille 9, la première période intéressante est 8. On remarque que pour la période 2, les deux sous-tableaux considérés sont déjà triés donc aucune inversion n'a lieu.*

9	8	7	6	5	4	3	2	1
2	8	7	6	5	4	3	9	1
2	1	7	6	5	4	3	9	8
2	1	7	6	5	4	3	9	8
2	1	7	3	5	4	6	9	8
2	1	7	3	5	4	6	9	8
2	1	4	3	5	7	6	9	8
2	1	4	3	5	7	6	9	8
1	2	3	4	5	6	7	8	9

3. Que remarquez-vous sur l'exemple précédent ?

*L'exemple précédent est un pire cas pour le tri par insertion : les éléments sont triés dans l'ordre décroissant. Le nombre d'inversion d'éléments effectués par le tri par insertion est donc de  $\frac{n(n-1)}{2}$  où  $n$  est le nombre d'éléments du tableau (ici on a un tableau à 9 éléments donc 36 inversions pour le tri par insertion). On voit sur l'exemple qu'avec les périodes  $2^k$  choisies, le tri de Shell ne fait que 8 inversions (dont 4 pour la dernière étape qui est un tri par insertion classique).*