# L3, année 2007-08
# Algorithmique et Programmation
**Contrôle du 21 janvier 2008**
**Notes de cours autorisées, à l'exclusion de tout autre document. Les calculatrices sont autorisées.**

INTRODUCTION
Soit $n$ entiers positifs premiers entre eux, $a = (a_1, \ldots, a_n)$. Un entier est dit *$a$-représentable* s'il peut s'écrire $v \cdot a$ pour un vecteur $v$ de $\mathbb{N}^n$, où $u \cdot v$ représente le produit scalaire des deux vecteurs $u$ et $v$. Déterminer la $a$-représentativité d'un entier est connu comme le problème d' *une instance du problème de Frobénius*. Le nombre de Frobénius $g(a_1, \ldots, a_n)$ est le plus grand entier qui n'est pas $a$-représentable. Déterminer ce nombre est un problème $NP$-difficile, qui est polynomial si $n$ est fixe. Nous nous intéressons au cas $n = 3$ dans la suite pour donner une borne sur le Shellsort meilleure que celle donnée dans le cours avancé page 11.

## 1 Généralités

1. Le problème de Frobénius est bien posé dans le cas où les entiers sont premiers entre eux.
   (a) Montrer comment déterminer un vecteur $v \in \mathbb{Z}^n$ tel que $v \cdot a = 1$. Quel est le coût algorithmique de votre solution ?
   (b) On note $m = \min_{i=1}^{n} v_i$ et on suppose que $a_1 < \ldots < a_n$. Considérons $u = a_1|m|(1, \ldots, 1)$, le vecteur de taille $n$ où toutes les composantes sont égales à $a_1|m|$. Montrer que les entiers $u + iv$ appartiennent à $\mathbb{N}$ pour $i = 0, \ldots, a_1 - 1$. Montrer qu'il existe $a_1$ entiers consécutifs qui sont $a$-représentable.
   (c) En déduire qu'il n'y a qu'un nombre fini d'entier $t \in \mathbb{N}$ qui n'est pas $a$-représentable.
2. Montrer que dans le cas $n = 2$, on peut montrer que $g(a_1, a_2) = (a_1 - 1)(a_2 - 1) - 1$.
3. Résoudre algorithmiquement la solution pour $n = 2$.

## 2 Algorithme pour le cas $n = 3$

## 3 Complexité du Shellsort

Soit $T$ un tableau de $N$ entiers, $T[1], \ldots, T[N]$.
La lenteur du tri par insertion est due au fait que les seules permutations réalisées le sont avec des éléments adjacents. Par exemple, si l'élément de plus petite valeur est à la fin du fichier, $N$ étapes sont nécessaires. Le tri Shell est une extension du tri par insertion qui entraîne un gain de temps en permettant des permutations d'éléments éventuellement très éloignés.

Une passe de cet algorithme de tri, aussi appelée $h$-tri, réorganise le tableau d'origine en plusieurs sous-tableaux contenant les éléments $T[1], T[h], T[2h], \ldots T[\lfloor N/h \rfloor]$ pour le premier sous-tableau, $T[2], T[h+1], T[2h+1]$, ... pour le deuxième sous-tableau, ... Ensuite sur chaque sous-tableau, on effectue un tri par insertion. L'algorithme consiste à effectuer plusieurs passes en utilisant des incréments par valeur décroissantes, $h_n > h_{n-1} > \ldots > h_1$.
Dans le cours avancé, vous avez vu que la complexité du tri Shell pouvait être en $O(N^{3/2})$ avec une suite d'incrément donnée page 18 et 19.
On suppose le lemme suivant démontré dans le cours avancé : Soit $k$ et $h$ deux incréments. En appliquant un $h$-tri à un tableau $k$-trié, on obtient un tableau qui est $h$ est $k$ trié.

1. Donner en pseudo-code l'algorithme du Shellsort.
2. On considère la suite d'incréments $1, 8, 23, 281, \ldots$ où $h_j = 4^{j+1} + 3 \times 2^j + 1$. Montrer que cette suite d'incréments vérifie les hypothèses du théorème de Selmer pour chaque triplet $h_{j+1}, h_{j+2}, h_{j+3}$.
3. En déduire que
$$g(h_{j+1}, h_{j+2}, h_{j+3}) = O(h_j^{3/2}).$$
4. En utilisant la même technique de preuve en découpant la complexité en deux que page 18/19, montrer que la complexité est en $O(N^{4/3})$. (Vous pouvez réutiliser le théorème 4 du cours avancé.)