Université de Limoges **Année Universitaire 2008-2009**
Faculté des Sciences et Techniques

Master Mathématiques première année
Algorithmique et programmation
Examen du 9 janvier 2009

Durée 2h. Documents manuscrits et distribués durant le semestre autorisés.
Les exercices sont indépendants. Le barême est indicatif.

Exercice 1. (6 points) **Tri par bulles.** On s'intéresse à l'algorithme suivant :

```
Algorithme Tri_bulle(var T: tableau d'entiers, n: entier)
Entrées : T[0],...,T[n-1], tableau de n entiers distincts.
Sortie : Le tableau trié en ordre croissant.
Début
    trié ⟵ faux
    m ⟵ n-1
    Tant que non trié faire
        trié ⟵ vrai
        Pour i de 0 à m-1 faire
            Si T[i] > T[i+1] alors
                tmp ⟵ T[i]
                T[i] ⟵ T[i+1]
                T[i+1] ⟵ tmp
                trié ⟵ faux
            fin
        fin
        m ⟵ m-1
    fin
Fin.
```

On s'intéresse à la complexité de cet algorithme, en *nombre de comparaisons d'éléments du tableau*.

1. Vérifier qu'à l'issue de chaque exécution de la boucle «`Pour`», l'élément `T[m]` est correctement placé. En déduire que l'algorithme fonctionne.

2. Quelle configuration d'entrée correspond au meilleur des cas de cet algorithme? Expliquez. Que peut-on dire de la complexité asymptotique dans le meilleur des cas ?

3. Quelle configuration d'entrée correspond au pire des cas ? Expliquez. Que peut-on dire de la complexité asymptotique dans le pire des cas ?

Exercice 2. (6 points) Soit $m, m_1, \ldots, m_r$ des entiers positifs non nuls tels que $m = \prod_{i=1}^{r} m_i$. Soit p un nombre premier et α un élément non nul de $\mathbb{Z}/p\mathbb{Z}$. Pour $1 \leq i \leq r$, notons n_i les entiers m/m_i. On cherche à calculer efficacement les r éléments $\alpha^{n_1}, \ldots, \alpha^{n_r}$ de $\mathbb{Z}/p\mathbb{Z}$.

1. On considère l'algorithme naïf suivant : on calcule les n_i puis successivement les α^{n_i} par exponentiation binaire (utilisant la décomposition de l'exposant en base 2).

   (a) Quelle est la complexité *en nombre de multiplications dans* $\mathbb{Z}/p\mathbb{Z}$ de cet algorithme (en fonction de r et de la taille de m) ?

   (b) On suppose maintenant que p est un entier multiprécision. Que peut-on dire de la complexité en nombre d'opérations machine de l'algorithme (tenir compte de la taille de l'entier p) ?

2. On s'intéresse ensuite à un algorithme de type «diviser pour régner» :

   Algorithme Expo($\beta, t, p_1, p_2, \ldots, p_{2^t}$) : ensemble d'éléments de $\mathbb{Z}/p\mathbb{Z}$
   Entrées :
   β : élément de $\mathbb{Z}/p\mathbb{Z}$.
   t : un entier positif.
   $p_1, p_2, \ldots, p_{2^t}$: 2^t entier positifs.
   Début
   Si $t = 0$ alors renvoyer($\{\beta\}$)
   $a \longleftarrow p_1 * p_2 * \cdots * p_{2^{t-1}}$
   $b \longleftarrow p_{2^{t-1}+1} * p_{2^{t-1}+2} * \cdots * p_{2^t}$
   Renvoyer(Expo$(\beta^a, t-1, p_{2^{t-1}+1}, \ldots, p_{2^t}) \cup$ Expo$(\beta^b, t-1, p_1, \ldots, p_{2^{t-1}})$)
   End.

   (a) Expliciter le fonctionnement de l'algorithme dans le cas $t = 3$.

   (b) On suppose dans la suite que $r = 2^k$. Montrer par récurrence que Expo$(\alpha, k, m_1, \ldots, m_r)$ renvoie $\alpha^{n_1}, \ldots, \alpha^{n_r}$.

   (c) On suppose que tous les m_i ont sensiblement la même taille : $T(m_i) = T(m)/r$. Que peut-on dire de la complexité de cet algorithme, en nombre de multiplications dans $\mathbb{Z}/p\mathbb{Z}$? (on l'étudiera en fonction de r et m).

Exercice 3. (8 points) **Le type de données abstrait File**. (Attention: il ne s'agit pas du mot anglais «file», qui lui, se traduit par «fichier»). Ce type est destiné à représenter des files d'attente fonctionnant sur le mode "premier arrivé, premier servi" (First In, First Out); penser à une queue à la caisse d'un supermarché[1].
Les primitives que l'on va considérer sont:

1. `EstVide(F)` : teste si la file est vide,
2. `Afficher(F)` : affiche les éléments de la file, en commençant par les éléments de tête (ceux qui sont entrés en premier),
3. `Défiler(F)` : ôte un élément en tête de la file `F`,
4. `Tête(F)` : renvoie la valeur de l'élément en tête de la file `F`, sans le défiler,
5. `Enfiler(x,F)` : ajoute l'élément `x` à la file `F`,
6. `Vider(F)` : supprime les éléments de la file `F`.

On souhaite représenter le type File à l'aide d'une liste chaînée. Pour cela, il est commode et efficace de conserver un pointeur vers l'élément de tête (le premier entré, et donc le premier qui sortira), ainsi qu'un pointeur vers l'élément de queue (le dernier à être entré dans la file). Ceci conduit aux définitions suivantes:

```
typedef struct cellule {
    int valeur;
    struct cellule *lien;
} Cellule;

typedef struct file {
    CELLULE *tete;
    CELLULE *queue;
} File;
```

Ecrire le code C correspondant aux primitives ci-dessus. Les champs d'une structure de type File seront amenés à être modifiés par certaines fonctions; *dans ce cas on passera par référence le paramètre désignant la file*. Plus précisémment, les fonctions auront les prototypes suivants:

```
int EstVide(File);
void Afficher(File);
int Tete(File);
void Defiler(File*);
void Enfiler(int, File*);
void Vider(File*);
```

[1] Pour information: le mode de fonctionnement d'une PILE est "dernier arrivé, premier servi" (Last In, First Out).