Université de Limoges
Faculté des Sciences et Techniques

Master 1 Informatique
Master 1 Mathématiques

Examen du 20 juin 2008
Calculabilité, complexité et évaluation de performances

Exercice 1. On considère un entier naturel n et on note $(n_0, ..., n_k) \in \{0,1\}^{k+1}$ sont écriture en binaire (i.e. $n = \sum_{i=0}^{k} n_i * 2^i$).

1. Donner le développement en binaire du quotient de la division par 2 de n, qu'on notera $\text{quo}_2(n)$.
2. Donner une machine de Turing sur l'alphabet $\{0, 1\}$ permettant d'implanter la fonction $\lambda n.(\text{quo}_2(n) + 1)$.

Exercice 2. Soit $L = (l_1, ..., l_K)$ une liste d'entiers naturels. On dit que L est trié par ordre croissant si $l_1 \leqslant l_2 \leqslant \cdots \leqslant l_K$. On note $\text{Add}(n, L)$ la procédure permettant d'insérer un élément au début d'une liste et ne renvoyant rien, $\text{Extract}(L)$ la procédure retournant le premier élément de la liste L et supprimant cet élément du début de la liste et $\text{empty}(L)$ la fonction retournant 1 si la liste est vide et 0 sinon. Enfin, on dispose de la procédure $\text{append}(L, L')$ permettant de concaténer deux listes $L = (l_1, ..., l_K)$ et $L' = (l'_1, ..., l'_{K'})$, L recevant le resultat de la concaténation $(l_1, ..., l_K, l'_1, ..., l'_{K'})$ et la procédure ne retournant rien. On note () la liste vide.

1. Donner un algorithme permettant de vérifier qu'une liste est triée. Le problème de décider si une liste est triée est-il NP ?
2. On considère l'algorithme suivant :

Algorithme `DivSort`
Input : A list $L = (l_1, ..., l_K)$
  Lo $\leftarrow$ (); Up $\leftarrow$ ();
  pivot $\leftarrow$ Extract(L);
  tant que Empty$(L) = 0$ faire
    tmp $\leftarrow$ Extract(L);
    si tmp $>$ pivot alors
      Add(tmp, Up);
    sinon
      Add(tmp, Lo);
    fin si;
  Add(pivot, Lo);
  Retourner append(`DivSort`(Lo), `DivSort`(Up));
output : la liste triée.

En constatant que dans la boucle « tant que » Up ne contient que des éléments plus grands que le pivot et que Lo ne contient que des éléments plus petit, montrer par récurrence que cet algorithme trie les listes d'entiers.

3. On suppose maintenant que $K = 2^k$ et qu'à la fin du while Lo et Up sont de taille $\frac{K}{2} = 2^{k-1}$ et qu'il en sera de même à chaque appel de la fonction `DivSort`. On compte comme opérations élémentaires les comparaisons d'entiers et les fonctions et procédures décritent sur les listes, sauf $\text{Append}(L, L')$ qui est linéaire en la taille de L. Donner une estimation de la complexité d'une exécution de cet algorithme avec ces hypothèses (si $K = 2^k$ est la taille de la liste à trier, vous devriez trouver $\mathcal{O}(K * k) = \mathcal{O}(K * \log_2(K))$ opérations élémentaires).

4. Le problème de trier une liste est-il polynômiale ?

Exercice 3. Soient n et m deux entiers de développement respectivement (n_0, ..., n_K) et $(m_0, ..., m_K)$ en base 2, i.e. $n = \sum_{i=0}^{K} n_i * 2^i$ et $m = \sum_{i=0}^{K} m_i * 2^i$. On rappelle que la multiplication de deux entiers de taille K se fait en $\mathcal{O}(K^2)$ opérations binaires par un algorithme naïf et que la somme et la différence se fait en $\mathcal{O}(K)$ opérations binaires. On suppose que $K = 2^k$, on a alors $n = \bar{n} * 2^{\frac{K}{2}} + \tilde{n}$ et $m = \bar{m} * 2^{\frac{K}{2}} + \tilde{m}$. On a $n * m = \bar{n} * \bar{m} * 2^K + (\bar{n} * \tilde{m} + \tilde{n} * \bar{m}) * 2^{\frac{K}{2}} + \tilde{n} * \tilde{m}$. On estime que la multiplication d'un entier x par 2^l est en $\mathcal{O}(\log_2(x) + l)$ opérations binaires. La décomposition précédente du produit ne donne pas un algorithme plus performant que le produit classique des entiers car il implique 4 produits d'entiers de taille moitier de la taille des entiers initiaux. Remarquons alors que $(\bar{n} + \tilde{n}) * (\bar{m} + \tilde{m}) = \bar{n} * \bar{m} + \bar{n} * \tilde{m} + \tilde{n} * \bar{m} + \tilde{n} * \tilde{m}$. En posant $A = \bar{n} * \bar{m}$, $B = \tilde{n} * \tilde{m}$ et $C = (\bar{n} + \tilde{n}) * (\bar{m} + \tilde{m})$, constater que $n * m = A * 2^K + (C - A - B) * 2^{\frac{K}{2}} + B$. . On considère alors l'algorithme suivant :

Algorithme `Karatsuba`

Entrée : $n = \sum_{i=0}^{K} n_i * 2^i = \bar{n} * 2^{\frac{K}{2}} + \tilde{n}$ et $m = \sum_{i=0}^{K} m_i * 2^i = \bar{m} * 2^{\frac{K}{2}} + \tilde{m}$

si $K = 0$ alors

retourner $n_0 * m_0$;

sinon

$A \leftarrow$ `Karatsuba`$(\bar{n}, \bar{m})$; $B \leftarrow$ `Karatsuba`$(\tilde{n}, \tilde{m})$; $C \leftarrow$ `Karatsuba`$((\bar{n} + \tilde{n}), (\bar{m} + \tilde{m}))$;

retourner $A * 2^K + (C - A - B) * 2^{\frac{K}{2}} + B$

fin si;

Sortie : $n * m$.

1. Montrer par récurrence que cet algorithme est correcte.
2. Avec les hypothèses de l'énoncé, donner la complexité de cet algorithme.