Algorithmes et Arbres, $2^{ème}$ Semestre 11/12

ALGORITHMES ET ARBRES

Durée : 3h, polycopiés et calculatrices autorisés, téléphones interdits.
Barême indicatif sur 20 : exo1(3 points), exo2(5 points), exo3(5 points), exo4(7 points).

Exercice 1. (Notations asymptotiques.)
On a la récurrence $T(1) = 1$ et $\forall n > 1, T(n) = n + T(\lfloor n/2 \rfloor)$.
1. Montrer que pour tout $k \geq 0$, $T(2^k) = 2^{k+1} - 1$.
2. En déduire que $T(n) = \Theta(n)$.

Exercice 2 (Insertion / suppression ABR). Répondre à chaque question en représentant seulement l'arbre obtenu sans justification.
1. Former un arbre binaire de recherche en insérant successivement, et dans cet ordre, les éléments :

$$12, 7, 3, 9, 14, 4, 13, 16, 1, 15, 10, 17.$$

2. Supprimer dans l'ordre les éléments 7, 14, 12.
3. Faire une rotation droite en 12.

Exercice 3. (Écrire les entiers uniquement avec des 1.) On cherche le nombre minimum $T(n)$ de un nécessaire pour "fabriquer" l'entier n en n'utilisant que des $+$, des $\times$ et des parenthèses. Exemple : $23 = ((1 + 1) \times (1 + 1 + 1) + 1) \times (1 + 1 + 1) + 1 + 1$ et $T(23) = 11$.
1. Pour n allant de 1 à 12, trouver $T(n)$ et donner une expression utilisant $T(n)$ un.
2. Trouver une récurrence de type programmation dynamique pour $T(n)$.
3. En déduire un $\mathcal{O}$ pour la complexité en nombre de comparaisons pour trouver $T(n)$.
4. Montrer que pour tout $n > 0$, $T(n + 1) \leq T(n) + 1$.
5. Montrer que pour tout $n > 0$, $T(2^n) = 2n$.

Exercice 4. Problème de la somme de sous-ensemble.
Considérons deux séquences d'entiers $x_1, ..., x_k$ et $y_1, ..., y_k$ et un entier cible C. On cherche un algorithme qui trouve (s'ils existent) x_i et y_j tel que $x_i + y_j = C$.
1. Donner le pseudo-code d'un algorithme en $\mathcal{O}(k \ln(k))$.

On a une séquence finie d'entiers S et un entier cible C. Le problème est de décider, s'il y existe une sous-séquence non-vide de S dont la somme des éléments vaut C. Exemple : avec $S = [1, 4, 9, 3, 8, 12]$ et $C = 18$ la réponse est oui, car 18=1+9+8. Un algorithme naïf $A0$ pour résoudre ce problème est de tester, pour tous les sous-ensembles non vides de S, si leur somme est égale à C.

2. Montrer que l'on peut calculer ces $2^{|S|} - 1$ sommes en faisant $2^k - k - 1$ additions.
3. En déduire la complexité en nombre d'additions de l'algorithme $A0$.
4. Donner un algorithme $A1$ plus efficace en séparant S en deux sous-ensembles de même taille et en utilisant la question 1.
5. Donner la complexité de l'algorithme $A1$.
6. On note S_i la séquence des i premiers éléments de S et $M(S_i, C)$ la plus grande somme plus petite que C faisable avec des éléments de S_i. Trouver une récurrence de type programmation dynamique semblable à celle du problème du sac à dos permettant de calculer $M(S, C)$.
7. Quelle est la complexité en nombre d'additions pour calculer $M(S, C)$?