Fiche TD 11

Exercice - Approximabilité de SUBSET-SUM

Dans un précédent TD, on s'est intéressé au problème de décision SUBSET-SUM consistant à savoir s'il existe un sous-ensemble d'entiers de S dont la somme vaut exactement t. Le problème d'optimisation qui lui est associé prend aussi en entrée un ensemble d'entiers strictement positifs S et un entier t et il consiste à trouver un sous-ensemble de S dont la somme est la plus grande possible sans dépasser t (cette somme qui doit donc approcher le plus possible t sera appelée *somme optimale*).

On suppose que $S = \{x_1, x_2, \ldots, x_n\}$ et que les ensembles sont manipulés sous forme de listes triées par ordre croissant. Pour une liste d'entiers L et un entier x, on note $L + x$ la liste d'entiers obtenue en ajoutant x à chaque entier de L. Pour listes L et L', on note **Fusion**(L, L') la liste contenant l'union des éléments des deux listes.

1 - *Premier algorithme*

Somme(S, t)

début
- $n \leftarrow |S|$;
- $L_0 \leftarrow \{0\}$;
- **pour** i **de** *1* **à** n **faire**
  - $L_i \leftarrow$ **Fusion**$(L_{i-1}, L_{i-1} + x_i)$;
  - Supprimer de L_i tout élément supérieur à t ;
- **retourner** le plus grand élément de L_n

fin

Quel est la distance entre la valeur retournée par cet algorithme et la somme optimale ? Quelle est la complexité de cet algorithme dans le cas général ? Et si pour un entier $k \geq 1$ fixé, on ne considère que les entrées telles que $t = \mathcal{O}(|S|^k)$, quelle est la complexité de cet algorithme ?

2 - *Deuxième algorithme*

Cet algorithme prend en entrée un paramètre ϵ en plus, où ϵ est un réel vérifiant $0 < \epsilon < 1$.

Somme-avec-seuillage(S, t)

début
- $n \leftarrow |S|$;
- $L_0 \leftarrow \{0\}$;
- **pour** i **de** *1* **à** n **faire**
  - $L_i \leftarrow$ **Fusion**$(L_{i-1}, L_{i-1} + x_i)$;
  - $L_i \leftarrow$ **Seuiller**$(L_i, \epsilon/2n)$;
  - Supprimer de L_i tout élément supérieur à t ;
- **retourner** le plus grand élément de L_n

fin

L'opération **Seuiller** décrite ci-dessous réduit une liste $L =< y_0, y_1, \ldots, y_m >$ (supposée triée par ordre croissant) avec le seuil δ :

```
Seuiller(L, δ)

début
    m ← |L| ;
    L' ←< y0 > ;
    dernier ← y0 ;
    pour i de 1 à m faire
        si yi > (1 + δ)dernier alors
            Insérer yi à la fin de L' ;
            dernier ← yi ;

    retourner L'
fin
```

Evaluer le nombre d'éléments dans L_i à la fin de la boucle. En déduire la complexité totale de l'algorithme. Pour donner la qualité de l'approximation fournie par cet algorithme, borner le ratio valeur retournée/somme optimale.