LICENCE MPCI

Programmation 2

.

Devoir surveillé

Les documents de cours sont autorisés. ***Tous les algorithmes doivent être prouvés et donnés avec leur complexité que l'on justifiera.***

1

1. Donnez les 4 types de complexités vus en cours.
2. Quels sont leurs intérêts respectifs ?

2

On considère l'algorithme suivant:

```
def Fusion (Liste1, Liste2) :
    n ← len(Liste1) + len(Liste2)
    p1 ← 0 ; p2 ← 0
    Liste1 ← Liste1 + [+∞]
    Liste2 ← Liste2 + [+∞]
    Resultat ← [ ]
    while len(Resultat) < n :
        if Liste1[p1] < Liste2[p2] :
            Resultat.append(Liste1[p1])
            p1 ← p1 + 1
        else :
            Resultat.append(Liste2[p2])
            p2 ← p2 + 1
    return Resultat
```

On rappelle que, appliqué à des listes, le + est la concaténation & que append ajoute un élément à la fin d'une liste.

1. Montrez que la complexité de cet algorithme est en $O(n)$
2. Montrez que si Liste1 & Liste2 sont triées, alors le résultat est trié & est constitué de tous les éléments de Liste1 & Liste2.
3. Que faudrait-il changer pour fusionner k listes (au lieu de 2) en $O(n \cdot k)$, où n est la somme des longueurs des listes.

3

1. En utilisant la question précédente, proposez un algorithme de tri récursif qui trie 2 parties d'une liste avant de les fusionner.

2. Peut-on faire mieux pour trier une liste ?

4

Étant donné un polynôme $\mathcal{P}(X) = \sum_{k=0}^{n} a_k \cdot X^k$ (par exemple $\mathcal{P}(X) = 4X^3 + 2X^2 + 7$) & un réel x (par exemple $x = 2$), évaluer $\mathcal{P}$ (en x) consiste à déterminer la valeur de $\mathcal{P}$ pour $X = x$ (ici, 47).

1. Proposez une façon de stocker un polynôme dans une liste.

2. Si l'on possède une fonction `puissance(x, n)` qui élève `x` à la puissance `n` en $\mathcal{O}(n)$ opérations, proposez un algorithme qui prend en entrée une liste représentant un polynôme et un réel et qui rend l'évaluation du polynôme.

3. En remarquant que $x^n = x * x^{n-1}$, proposez un algorithme linéaire pour évaluer un polynôme.

5

Soient deux chaînes de caractères S_1 et S_2. On dit que S_2 est un *sous-mot* de S_1 s'il existe un indice i tel que $S_2[j] = S_1[i + j]$ pour tout j de 0 à $len(S_2) - 1$.

Proposez un algorithme qui détermine si S_2 est un sous-mot de S_1.