Exercice 0. Notations asymptotiques

Pour chaque fonction f donner une fonction plus simple g telle que $f(n) = \Theta(g(n))$. Justifier

1. $f(n) = n + n\log n + \sqrt{n}$

   Prenons $g(n) = n\log n$, pour $n \to \infty$

$$\left|\frac{f(n)}{g(n)}\right| = \underbrace{\left|\frac{n}{n\log n}\right|}_{\to 0} + \left|\frac{n\log n}{n\log n}\right| + \underbrace{\left|\frac{\sqrt{n}}{n\log n}\right|}_{\to 0}$$

   alors $\left|\frac{f(n)}{g(n)}\right| \longrightarrow 1$, en particulier $\left|\frac{f(n)}{g(n)}\right|$ est bornée.

   On a aussi

$$\left|\frac{g(n)}{f(n)}\right| = \left|\frac{n\log n}{n + n\log n + \sqrt{n}}\right| \leq 1$$

   D'où $n + n\log n + \sqrt{n} \in \Theta(n\log n)$ quand $n \to \infty$.

2. $f(n) = (0.99)^n + n^{100}$

   Comme $\lim_{n\to\infty}(0.99)^n = 0$. Prenons $g(n) = n^{100}$, alors $\left|\frac{f(n)}{g(n)}\right| = \left|\frac{(0.99)^n + n^{100}}{n^{100}}\right| \xrightarrow[n\to\infty]{} 1$, en particulier $\left|\frac{f(n)}{g(n)}\right|$ est bornée.

   On a aussi

$$\left|\frac{g(n)}{f(n)}\right| = \frac{n^{100}}{(0.99)^n + n^{100}} \leq 1$$

   D'où $(0.99)^n + n^{100} \in \Theta(n^{100})$ quand $n \to \infty$.

3. $f(n) = (1000)2^n + 4^n$

   Prenons $g(n) = 4^n$, alors $\frac{f(n)}{g(n)} \xrightarrow[n\to\infty]{} 1$. Donc $\left|\frac{f(n)}{g(n)}\right|$ est bornée.

   On a aussi $\left|\frac{4^n}{(1000)2^n + 4^n}\right| \leq 1$. D'où $(1000)2^n + 4^n \in \Theta(4^n)$ quand $n \to \infty$.

4. $f(n) = \sum_{k=1}^{n} 2.4^k = 8 + 32 + \ldots + 2.4^n$

$$\begin{aligned} f(n) &= 2\sum_{k=1}^{n} 4^k \\ &= 2\left(\frac{4^{n+1} - 4}{3}\right) \\ f(n) &\in \Theta(4^n) \end{aligned}$$

5. $f(n) = \sum_{k=0}^{n} 2k^2 = 2.1^2 + 2.2^2 + 2.3^2 + \ldots + 2.n^2$

Méthode 1 : majorations (on peut prendre la partie entière $\frac{n}{2}$)

$$\begin{aligned} \sum_{k=0}^{n} k^2 &\geq \left(\frac{n}{2}\right)^2 + \left(\frac{n}{2}+1\right)^2 + \ldots n^2 \\ &\geq \left(\frac{n}{2}\right)^2 \frac{n}{2} && \text{min} \times \text{nombre de termes} \\ &\geq \frac{n^3}{8} \end{aligned}$$

et

$$\sum_{k=1}^{n} k^2 \leq nn^2 \qquad \text{max} \times \text{nombre de termes}$$

d'où $\sum_{k=0}^{n} 2k^2 = \Theta(n^3)$.

Méthode 2: Formule exacte

$$\begin{aligned} \sum_{k=0}^{n} 2k^2 &= 2\sum_{k=0}^{n} k^2 \\ &= 2\frac{n(n+1)(2n+1)}{2} \end{aligned}$$

d'où $\sum_{k=0}^{n} 2k^2 = \Theta(n^3)$.

Problème 1. Tableau unimodal

Dans un tableau unimodal, les éléments sont triés dans l'ordre croissante jusqu'à un maximum, ensuite les éléments sont triés dans l'ordre décroissante.

Par exemple, le tableau $[1, 6, 14, 25, 37, 42, 18, 7, -3]$ est unimodal avec 42 le maximum.

Écrire en pseudocode un algorithme recursif MAX(U) pour trouver le maximum dans un tableau unimodal U à n éléments distincts. Calculer la complexité de votre algorithme.

1. Algorithme en $O(n)$

Un algorithme de recherche de maximum pour un tableau quelconque (recursif ou iteratif) a pour complexité $\Theta(n)$.

2. Algorithme en $O(\log n)$

Dans un tableau unimodal, il faut remarquer qu'il y a un seul sommet (supérieur à ses voisins de gauche et de droite), qui est le maximum. On peut utiliser un algorithme dichotomique pour le retrouver.

```python
import math
def maxu(U):
    n = len(U)
    if n == 0:
        M = None
    elif n == 1:
        M = U[0]
    elif n == 2:
        if U[0]<U[1]:
            M = U[1]
        else:
            M = U[0]
    else:
        mil = math.floor(n/2)
        if U[mil-1] < U[mil] and U[mil] > U[mil+1]:
            M = U[mil]
        elif U[mil-1] < U[mil] < U[mil+1]:
            M = maxu(U[mil+1:])
        elif U[mil-1] > U[mil] > U[mil+1]:
            M = maxu(U[:mil])
    return M
```

Problème 2. Proche paire

Écrire un algorithme pour trouver dans un tableau la paire de nombres qui sont les plus rapprochés l'un de l'autre.

Par exemple, dans le tableau $[-3, 1, -8, 7, 10, 3, 14]$ la paire de nombres les plus rapprochés est $(1, 3)$. Calculer la complexité de votre algorithme.

La solution la plus facile utilise un "brute force" algorithme, sa complexité est $O(n^2)$. Il faut que $len(L) \geq 2$.

```python
def ppaire(L):
    paire = (L[0],L[1])
    d = abs(L[0]-L[1])
    n = len(L)
    for i in range(n):
        for j in range(n):
            if i!=j and abs(L[i]-L[j]) < d:
                paire = (L[i],L[j])
                d = abs(L[i]-L[j])
    return paire
```

Une autre façon est d'arranger la liste par ordre croissante avant de procéder. La complexité peut descendre jusqu'à $O(n \log n)$ selon l'algorithme de triage utlisé.

Problème 4. Tri à bulle

Le tri à bulle fait le tri croissant d'un tableau L (avec $n = len(L)$) de la façon suivante

1. On compare $L[0]$ et $L[1]$. Si $L[1] > L[0]$ alors on échange $L[0]$ et $L[1]$. Ensuiter on fait de même pour $L[1]$ et $L[2]$, et ainsi de suite jusqu'à $L[n-2]$ et $L[n-1]$. Écrire le pseudocode ou code Python pour faire cette opération (boucle intérieure).

   ```
   for i in range(0,n):
       if L[i] > L[i+1]:
           L[i],L[i+1] = L[i+1],L[i]
   ```

2. Qu'est ce qu'on obtient après cette opération.

   Exemple: le plus grand élément se trouve dans L[n−1]

3. Pour faire le tri complet de L on doit répéter cette opération un certain nombre de fois (boucle extérieur). Ecrire l'algorithme pour faire le tri complet.

   ```
   for k in range(1,n):
       for i in range(0,n-k):
           if L[i] > L[i+1]:
               L[i],L[i+1] = L[i+1],L[i]
   ```

4. Determiner un invariant pour la boucle extérieure et calculer la complexité de l'algorithme.

   - Après la boucle $k = 1$, $L[n-1]$ contient le plus grand élément.
   - Après la boucle $k = 2$, $L[n-2]$ contient le deuxième plus grand élément.
   - ...

   Un invariant: à l'étape k, $L[n-k:]$ contient les k plus grands éléments dans l'ordre croissante.