

## Interrogation 4 (2 décembre 2024)

---

durée : 2 heures

Pour chacun des algorithmes, on justifiera avec soin :

- que l'algorithme termine.
- que l'algorithme renvoie bien le résultat demandé.

On demande des algorithmes en pseudo-code, mais pour simplifier la rédaction, on peut utiliser certaines syntaxes du langage Python comme les opérateurs `//` et `%` sur les entiers, les opérateurs `+` (concaténation) et `len(L)` sur les listes ou la sélection de sous-listes comme `L[2 : 5]`. On évitera d'utiliser des fonctionnalités trop avancées de python, surtout celles non vues en TD/TP : on rédigera en pseudo-code avec des techniques élémentaires. En cas de doute, poser simplement la question ou rédiger de façon détaillée.

Pour les boucles « pour », on adoptera les conventions suivantes. Si  $a, b \in \mathbb{Z}$ , « pour  $i$  allant de  $a$  à  $b$  » signifie « pour  $i$  parcourant en croissant l'intervalle  $\llbracket a, b \rrbracket$  ». Lorsque  $b < a$ , cet intervalle est vide, donc aucune des instructions dans la boucle « pour » n'est effectuée (ce sera par exemple le cas si on écrit « pour  $i$  allant de 1 à  $n$  », avec  $n = 0$ ). On n'utilisera pas d'autre type de boucles « pour » (décroissantes, saut d'indice etc) : dans ces situations, on utilisera l'instruction « tant que ».

Dans cette interrogation, « complexité » signifie « complexité en temps », on ne demandera pas de complexité en espace. Les complexités sont entendues dans le pire des cas.

**Exercice 1.** 1. Déterminer un équivalent en  $+\infty$  de  $\ln(n^2 + n + 1) - \ln(n^2 + n - 1)$ .  
2. Déterminer un équivalent en  $+\infty$  de  $\left(\frac{n}{n+1}\right)^{n^2}$ .

**Exercice 2.** 1. Écrire une fonction récursive (sans boucle) `est_dans_liste(L,e)` qui prend en entrée une liste  $L$  et un élément  $e$  et qui retourne le booléen `Vrai` si  $e$  est un des éléments de la liste  $L$  et `Faux` sinon. (Remarque : en Python, on utiliserait l'opérateur « `in` », en écrivant par exemple `monBooleen = e in L`. On demande de réimplémenter cette fonctionnalité à la main.)  
2. Écrire une fonction récursive (sans boucle) `est_dans_sous_liste(L,e)` prenant en entrée une liste  $L$  de listes d'entiers ainsi qu'un entier  $e$  et retournant le booléen `Vrai` si  $e$  appartient à l'une des sous-listes, et `Faux` sinon. On peut utiliser la première question.

Exemples : `est_dans_liste([1,3,2],3)` renvoie `Vrai`, `est_dans_liste([[1,3,2],[5,4]],4)` renvoie `Faux`, `est_dans_sous_liste([[1,3,2],[5,4]],4)` renvoie `Vrai`.

**Exercice 3.** (Aplatissement) Écrire une fonction récursive `aplatir(L)` prenant en entrée une liste  $L$  contenant des listes d'entiers, et retournant la concaténation de toutes ces listes d'entiers. Par exemple `aplatir([[1,2],[7],[3,5]])` retourne la liste `[1,2,7,3,5]`. La fonction ne doit pas utiliser de boucle, uniquement de la récursivité.

**Exercice 4.** On se donne une fonction `fusion(L1,L2,L3)` qui prend en entrée trois listes de nombres triés par ordre croissant, et qui retourne une unique liste de nombres triés par ordre croissant, résultant de la fusion des trois listes. On suppose que cette fonction fusionne les listes avec une complexité (temporelle) de  $O(p + q + r)$  avec  $p, q$  et  $r$  les longueurs des trois listes. Si les trois listes ont la même longueur  $n$ , la complexité est donc en  $O(n)$ .

On considère la variation suivante du tri fusion : on reçoit en entrée une liste non triée, on la coupe en trois (au lieu de deux), on trie récursivement les trois listes puis on les refusionne et on retourne la liste triée obtenue.

Comparer la complexité de cet algorithme de tri avec le tri fusion classique.

**Exercice 5.** *Dans cet exercice, on considère des nombres entiers de grande taille, où la taille d'un nombre désignera ici son nombre de chiffres en base 10. (On rédigera tout en base 10.)*

*Présenter l'algorithme de multiplication rapide de Karatsuba et calculer sa complexité, en fonction de la taille des nombres donnés en entrée.*

*(Rappel : il s'agit de l'algorithme vu en cours qui multiplie deux nombres de longueur  $n$  en complexité strictement meilleure que  $n^2$ . Pour calculer la complexité, on supposera que l'on peut additionner des nombres de taille  $n$  en temps  $O(n)$ . La multiplication par une puissance de 10 correspond à un décalage des chiffres et on suppose ici quelle s'effectue en  $O(n)$ .)*

**Exercice 6 (Bonus).** *Implémenter la variation du tri fusion introduite dans l'exercice 4. (La fusion de trois listes triées, et l'algorithme global.)*

## Correction succincte

### Correction de l'exercice 1

1. On a

$$\begin{aligned}\ln(n^2 + n + 1) - \ln(n^2 + n - 1) &= \ln\left(\frac{n^2 + n + 1}{n^2 + n - 1}\right) \\ &= \ln\left(1 + \frac{2}{n^2 + n - 1}\right) \\ &\sim \frac{2}{n^2 + n - 1} \\ &\sim \frac{2}{n^2}\end{aligned}$$

On peut aussi faire un DL de chacun des deux termes  $\ln\left(1 + \frac{1}{n} \pm \frac{1}{n^2}\right)$  à la précision  $o(1/n^2)$ .

2. On écrit

$$\left(\frac{n}{n+1}\right)^{n^2} = \exp\left(n^2 \ln \frac{n}{n+1}\right).$$

On effectue donc un DL de  $\ln \frac{n}{n+1}$  :

$$\begin{aligned}\ln \frac{n}{n+1} &= \ln\left(1 - \frac{1}{n+1}\right) = -\frac{1}{n+1} - \frac{1}{2}\left(\frac{1}{n+1}\right)^2 + o\left(\frac{1}{n^2}\right) \\ &= -\frac{1}{n} + \frac{1}{n(n+1)} - \frac{1}{2}\left(\frac{1}{n+1}\right)^2 + o\left(\frac{1}{n^2}\right) \\ &= -\frac{1}{n} + \frac{1}{2n^2} + o\left(\frac{1}{n^2}\right)\end{aligned}$$

On en déduit que  $n^2 \ln \frac{n}{n+1} = -n + \frac{1}{2} + o(1)$  et donc que

$$u_n = \exp\left(-n + \frac{1}{2} + o(1)\right) = e^{-n} e^{1/2} e^{o(1)} \sim e^{-n} e^{1/2}$$

### Correction de l'exercice 2

```
1. def est_dans_liste(L,e):
    if L == []:
        return False
    return L[0]==e or est_dans_liste(L[1:],e)
```

Preuve de terminaison : les appels récursifs se font sur des listes ayant à chaque fois un élément de moins et la récursion s'arrête pour une liste vide.

Autre rédaction : on montre par récurrence sur  $n \in \mathbb{N}$  que pour toute liste  $L$  de longueur  $n$ , `est_dans_liste(L,e)` produit  $n$  autres appels récursifs exactement.

Preuve de correction : on montre par récurrence que pour tout entier  $n$ , pour toute liste  $L$  de longueur  $n$ , pour tout objet  $e$ , `est_dans_liste(L,e)` renvoie le résultat voulu : Vrai si  $e$  est un élément de  $L$  et Faux sinon.

```
2. def est_dans_sous_liste(L,e):
    if L == []:
        return False
    return est_dans_liste(L[0],e) or est_dans_sous_liste(L[1:],e)
```

### Correction de l'exercice 3

```
def aplatir(L:list[list[int]]) -> list[int] :
    if L == []:
        return []
    return L[0] + aplatir(L[1:])
```

### Correction de l'exercice 4

Si  $n$  est multiple de trois, la complexité  $T(n)$  de cet algorithme de tri va vérifier

$$T(n) = 3T(n/3) + O(n)$$

En appliquant le « master theorem », on voit que la complexité obtenue est en  $O(n \log n)$ , donc du même ordre de grandeur que le tri fusion classique. On ne gagne donc pas d'ordre de grandeur de complexité en divisant en trois plutôt qu'en deux, dans ce cas-là.

### Correction de l'exercice 5

Définition de l'algo : cours. On coupe en deux et on effectue trois multiplications (au lieu de quatre), sur des nombres deux fois plus petits. (Le nombre d'additions et de décalages, lui, augmente, mais il est asymptotiquement négligeable par rapport au temps d'exécution des multiplications.)

Si  $n$  est pair, la complexité  $T$  va vérifier une relation de la forme

$$T(n) = 3T(n/2) + O(n)$$

D'après le « master theorem », comme  $\log_2(3) > 1$ , on a alors  $T(n) = O(n^{\log_2(3)})$ .

### Correction de l'exercice 6

On peut utiliser l'algorithme de fusion de deux listes et l'appliquer deux fois :

```
def fusion2(L1,L2):
    if L1 == []:
        return L2
    if L2 == []:
        return L1
    if L1[0] <= L2[0]:
        return [L1[0]] + fusion2(L1[1:],L2)
    return [L2[0]] + fusion2(L1,L2[1:])

def fusion3(L1,L2,L3):
    return fusion2(fusion2(L1,L2),L3)
```

Ensuite, l'algorithme de tri :

```
def tri(L):
    n = len(L)
    if n <= 1:
        return L
    if n == 2 and L[0] <= L[1]:
        return L
    if n == 2 and L[0] > L[1]:
        return [L[1],L[0]]
    return fusion3(tri(L[:n//3]),tri(L[n//3:2*(n//3)]),tri(L[2*(n//3):]))
```

On a rajouté le cas de longueur deux car sinon, lors de la découpe en trois parties, une liste de longueur deux serait coupée en deux listes vides et une liste de longueur deux, provoquant une récursivité infinie.