TD 4 : Complexité des algorithmes

En cours, on a commencé à voir comment calculer la complexité remporelle des algorithmes : on détermine quelles opérations sont élémentaires, on les compte en fonction de la taille de l'entrée, et on l'exprime comme un grand-$\mathcal{O}$ suffisamment fin. On va voir d'abord une première façon de l'évaluer en exprimant la complexité avec une relation de récurrence : cela sera souvent la méthode naturelle pour les fonctions récursives.

Exercice 1 – Complexité d'algorithmes classiques

On reprend le code du tri par insertion sur les listes en OCaml (que vous avez noté quelque part et ramené en TD, évidemment).

▶ **Question 1** Déterminer une relation de récurrence vérifiée par la complexité temporelle de `insere`. On considèrera les comparaisons comme des opérations élémentaires.

▶ **Question 2** Déterminer formellement la complexité temporelle au pire et au mieux de la fonction `insere`.

▶ **Question 3** Soit u une suite réelle en $\mathcal{O}(n)$. Montrer que $v : k \mapsto \sum_{i=0}^{k} u_k = \mathcal{O}(n^2)$. Qu'obtiens-t-on si $u = \mathcal{O}(1)$?

▶ **Question 4** En déduire les complexités au pire et au mieux de `tri_insertion`.

Considérons maintenant que les comparaisons ne sont pas des opérations élémentaires et notons $m(l)$ un majorant de la complexité temporelle des comparaisons entre les éléments de `l`.

▶ **Question 5** Rappeler le nombre de comparaisons des fonctions `insere` et `tri_insertion`, dans le pire et dans le meilleur cas. Exprimer alors la complexité temporelle de `insere` et `tri_insertion` en fonction de $m(l)$ et $n = |l|$.

▶ **Question 6** Déterminer la complexité temporelle de la fonction `tri_fusion`.

▶ **Question 7** Déterminer les complexités spatiales de `tri_insertion` et `tri_fusion`.

Exercice 2 – Étude de l'algorithme d'Euclide

L'algorithme d'Euclide calcule le PGCD de deux entiers naturels. Il s'écrit simplement :

OCaml
```ocaml
let rec pgcd a b =
  if b = 0 then a else pgcd b (a mod b)
```

On supposera dans la suite que $a \geq 0$ et $b \geq 0$. On admettra la correction partielle de cet algorithme (preuve en cours de maths 😅). On va étudier le nombre de divisions (dans ce cas, le nombre de fois qu'on appelle **`mod`**) effectuées par cette fonction lorsqu'elle est appelée sur a, b, que l'on notera $f(a, b)$.

▶ **Question 1** Démontrer la terminaison de cet algorithme et borner $f(a, b)$ en fonction de b.

▶ **Question 2** Écrire une fonction `etapes : int -> int -> int` telle que `etapes a b` calcule $f(a, b)$.

Soit $\phi : n \in \mathbb{N}^* \mapsto \max_{0 \leq k < n} f(n, k)$. On définit la suite $(F_n)_{n \in \mathbb{N}}$ par récurrence double :

$$\begin{aligned} F_0 &= 1 \\ F_1 &= 2 \\ F_{n+2} &= F_n + F_{n+1} \quad \text{pour } k \geq 2 \end{aligned}$$

▶ **Question 3** Déterminer $f(F_{n+1}, F_n)$ pour $n \in \mathbb{N}$.

▶ **Question 4** Montrer que pour tout $n \in \mathbb{N}$, si $1 \leq a < F_n$, alors $\phi(a) < n$.

▶ **Question 5** Montrer que pour tout $n \in \mathbb{N}$, $F_n \geq \left(\frac{3}{2}\right)^n$.

▶ **Question 6** En déduire que $\phi(a) = \mathcal{O}(\ln a)$, *c'est-à-dire il existe* $A > 0$ *tel que pour tout* $a \in \mathbb{N}$, $\phi(a) \leq A \ln a$ *à partir d'un certain rang (ici* 0 *suffit).*

▶ **Question 7** Que peut-on en conclure sur la complexité temporelle de l'algorithme d'Euclide ?

Exercice 3 – Recherche dichotomique

On reprend les algorithmes classiques du cours et des TD/TPs.

Pseudo-code

```
fonction RECHERCHE(x, t) :
    deb, fin ← 0, n
    Tant que fin − deb > 0, faire :
        milieu ← (deb + fin)/2                ▷ division entière
        Si t_milieu = x alors
            Renvoyer milieu
        Sinon, si t_milieu < x
            deb ← milieu + 1
        Sinon
            fin ← milieu
        Fin si
    Fin tant que
    Renvoyer n
Fin fonction
```

▶ **Question 1** Déterminer la complexité de l'algorithme de recherche dichotomique ci-dessus. On ne comptera que les comparaisons.

▶ **Question 2** Pourquoi compter les comparaisons suffit ?

Exercice 4 – Maxima de tranches

On travaille sur un tableau $t = [t_0, \dots, t_{n-1}]$. On cherche à calculer les *maxima par tranche de h* du tableau, définis comme suis pour $i \in [\![0, n-h]\!]$:

$$m_h(i) = \max(t\,[i\ :\ i+h])$$

▶ **Question 1** Écrire un algorithme naïf qui calcule ce tableau. On pourra s'aider d'un algorithme auxiliaire MAXTRANCHE(t, i, j) qui calcule le maximum de la tranche $t\,[i\ :\ j]$.

▶ **Question 2** Déterminer la complexité de cet algorithme.

▶ **Question 3** Il est possible d'écrire un algorithme de complexité en $\mathcal{O}(n)$ pour effectuer cette opération. Proposer une stratégie pour le faire.

▶ **Question 4** (À la maison) Implémenter cette stratégie en C.

Exercice 5 – Calcul de suite rapide

Le but de cet exercice est de calculer la suite définie par récurrence :

$$\begin{cases} u_0 = 1 \\ u_n = \dfrac{u_{n-1}}{1} + \dfrac{u_{n-2}}{2} + \dots + \dfrac{u_0}{n} \quad \text{pour } n > 0 \end{cases}$$

▶ **Question 1** Implémenter cette suite en une fonction récursive `double u(int n)` traduisant directement la relation de récurrence.

▶ **Question 2** En comptant uniquement les divisions, déterminer une relation de récurrence vérifiée par la complexité temporelle de `suite`.

▶ **Question 3** Proposer une version améliorée de votre fonction de complexité linéaire. Est-ce que vous y voyez un désavantage par rapport à la première ?

Exercice 6 – Encore de la complexité

On considère la fonction suivante :

OCaml

```ocaml
let rec f = function
  | [] -> []
  | h :: t -> h + List.length t :: f t
```

▶ **Question 1** Déterminer la spécification de la fonction `f`.

▶ **Question 2** Déterminer la complexité temporelle et spatiale de `f`.

▶ **Question 3** Proposer une fonction de complexité spatiale et temporelle linéaire répondant à la même spécification que `f`. Montrer sa correction.