Corrigé du DS2

Exercice 1 Questions de cours

1. Soient $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ deux suites à valeurs réelles. Donner la définition de $u_n = O(v_n)(n \to +\infty)$.
   La définition est $\exists n_0 \in \mathbb{N}, \exists A \in \mathbb{R}_*^+, \forall n \geq n_0, |u_n| \leq A|v_n|$
2. On somme les opérations effectuées par la boucle interne (celle en j) : $\sum_{j=i}^{n-1} 2 = 2(n-1-i+1)$. Il y a deux opérations car on fait une multiplication et une addition.
   On somme les opérations effectuées par la boucle externe (celle en i) :

$$\begin{aligned}\sum_{i=0}^{n-1} 2(n-1-i+1) + 2 &= 2n^2 - 2\frac{n(n-1)}{2} + 2 * n \\ &= n^2 + n \\ &= \Theta(n^2)\end{aligned}$$

   Pour la deuxième fonction même principe. La boucle interne fait $i+4-i+1 = 5$ itérations et à chaque itération effectue 3 additions. Soit au total 15 additions.
   La boucle externe effectue $4-0+1 = 5$ itérations avec à chaque fois 15 additions pour la boucle interne et 2 en plus. Soit un total de 85 additions.
   La complexité de cette fonction est un $O(1)$.
3. Alice, Bob et Charlie ont chacun écrit un algorithme pour résoudre leur DM. Celui d'Alice a une complexité en $O(n\log(n))$, celui de Bob a une complexité en $O(n!)$ et celui de Charlie a une complexité en $O(n)$. Qui a écrit l'algorithme le plus efficace ? Le moins efficace ?
   D'après le tableau de comparaison qu'on a écrit en cours, n est meilleur que $n\log(n)$ qui est meilleur que $n!$. ($n!$ étant pire que n'importe quelle exponentielle.)
4. Écrire une fonction Ocaml `length : 'a list -> int` qui calcule la taille d'une liste d'entiers.

```
let rec length l = match l with
  |[] -> 0
  |h::q -> 1+length q;;
```

5. Déterminer le type des fonctions Ocaml suivantes :

```
let f x y = x +. float_of_int y;;

f : float -> int -> float

let g (x,y,z) = if z = 'a' then x*y/2
                else 34;;

g : int * int * char -> int

let h x y z = if x<y then z-.5.
              else 4.;;

h : 'a -> 'a -> float -> float

let i x y z a = if x<y then z
                else a;;

i : 'a -> 'a -> 'b -> 'b -> 'b
```

Exercice 2 Quelques fonctions en Ocaml

1.
```
let f1 x = let y = log (x-.3.) +. 5. in y/.3.;;
```

2.
```
let rec f2 n =
  if n=0 then 0
  else let un = f2 (n-1) in un*un+5;;
```

3.
```
let f3 () =
  let rec aux n =
    if f2 n > 55 then n
    else aux (n+1)
  in aux 0;;
```

4.
```
let rec f4 n = match n with
  |0 -> 0
  |_ -> 3*n*n*n*n-2*n + f4 (n-1);;
```

5.
```
let h x (a,b) l = if x = a then (x,List.hd l)
                  else if b = 0.5 then (0,List.hd l)
                  else (a,List.hd l);;
```

Exercice 3 Un calcul de complexité récursive

L'algorithme **récursif** suivant propose de calculer le maximum d'un tableau, quand on considère uniquement les indices entre a et b (qui doivent être des indices valides du tableau).

Par exemple pour $tab = [1, 2, 4, 6, 1, 12]$ et $a = 1$ et $b = 4$, la fonction calcule le maximum du sous-tableau $[2, 4, 6, 1]$, soit 6.

Pour ce faire il découpe en deux le tableau, calcule le maximum de la moitié gauche m_1, le maximum de la moitié droite m_2 et renvoie le plus grand des deux (qui est le maximum du tableau entier)

```
int lemax(int a, int b, int* tab){
 if (a==b) {return tab[a];} //Le maximum d'un tableau de taille 1 est son unique élément
 else {
   int m1 = lemax(a,(a+b)/2, tab);
   int m2 = lemax((a+b)/2+1,b,tab);
   if (m1>m2){return m1;}
   else {return m2;}
 }
}
```

On note $C(n)$ la complexité de cette fonction pour des entrées telles que $n = b - a + 1$.

1. Le tableau entre les indices a et $(a + b)/2$ est de taille $\lceil n/2 \rceil$. L'autre tableau est de taille $\lfloor n/2 \rfloor$.
   Pour le montrer proprement, on peut séparer le cas où n est pair du cas où n est impair.
2. La suite $C(n)$ vérifie la relation de récurrence suivante : $C(1) = 1$ et $C(n) = C(\lceil n/2 \rceil) + C(\lfloor n/2 \rfloor) + 7$. Pour obtenir cette constante on a compté toutes les opérations algébriques et les tests réalisés.
   On suppose maintenant que $n = 2^k$ avec $k \in \mathbb{N}$. Cela nous permet d'ignorer les parties entières dans la suite du raisonnement.

   On montre par récurrence que pour $l \in [0, k]$, $C(n) = 2^l C(n/2^l) + 7 \sum_{i=0}^{l-1} 2^i$.

   En effet pour $l = 0$, on a $C(n) = 1 * C(n) + 0$.

   Supposons que $C(n) = 2^l C(n/2^l) + 7 \sum_{i=0}^{l-1} 2^i$ pour un $l \in [0, k]$ et essayons de montrer la propriété pour $l + 1$.

   En utilisant la formule de récurrence de C, on peut réécrire $C(n/2^l)$ en $2C(n/2^{l+1}) + 7$.

   En combinant les deux égalités, on obtient $C(n) = 2^l * (2C(n/2^{l+1}) + 7) + 7 \sum_{i=0}^{l-1} 2^i = 2^{l+1} C(n/2^{l+1}) + 2^l * 7 + 7 \sum_{i=0}^{l-1} 2^i =$
   $2^{l+1} C(n/2^{l+1}) + 7 \sum_{i=0}^{l+1-1} 2^i$.

   Par principe de récurrence la formule est vraie à tous les rangs, et en particulier au rang k. On a donc $C(n) =$
   $2^k C(1) + 7 * \sum_{i=0}^{k-1} 2^i = 2^k + 7 * (2^k - 1) = 8 * 2^k - 7 = 8 * n - 7 = \Theta(n)$.

Exercice 4 Preuve de programme : le retour d'Euclide

Pour chaque variable x on notera x_0 sa valeur avant la boucle et x_i sa valeur à la fin du i-ème tour de boucle.

1. Montrer la terminaison de cet algorithme sur toutes les entrées vérifiant la précondition.
   Montrons que b est un variant de la boucle while. Initialement b est positif et à chaque tour de boucle sa nouvelle valeur est $b' = a\%b$ qui vérifie $0 \leq b' < b$.
   Ainsi b est une quantité positive strictement décroissante.
   La boucle termine car elle admet un variant et l'appel récursif ne sert qu'à échanger a et b, donc sera fait au plus une fois. La fonctionne termine donc sur toutes les entrées valides.

2. Montrer que la propriété "à l'itération i, $PGCD(a_0, b_0) = PGCD(a_i, b_i)$" est un invariant.
   Supposons la propriété vraie à la fin de la i-ème itération. Elle l'est toujours au début de la $i+1$-ème. On a donc $PGCD(a_0, b_0) = PGCD(a_i, b_i)$.
   Lors de l'itération $i+1$, les nouvelles valeurs de a et b sont $a_{i+1} = b_i$ et $b_{i+1} = a_i \% b_i$. Or on sait que le pgcd de tous $c \leq d$ est le même que celui de d et $c\%d$.
   Donc $PGCD(a_{i+1}, b_{i+1}) = PGCD(b_i, a_i\%b_i) = PGCD(a_i, b_i) = PGCD(a_0, b_0)$. La propriété est bien invariante.
3. Conclure que la fonction renvoie bien le PGCD de a et b. La propriété est vraie à la première itération, en effet $PGCD(a_0, b_0) = PGCD(b_0, a_0\%b_0) = PGCD(a_1, b_1)$.
   Comme elle est invariante et que le boucle termine, on en déduit qu'elle est vraie après la boucle (on notera k le numéro de l'itération finale), soit lorsque $b_k = 0$. On a alors $PGCD(a_0, b_0) = PGCD(a_k, 0) = a_k$ par définition du PGCD. La valeur de renvoi de la fonction, qui est a_k, est donc bien le PGCD de a_0 et b_0.

Exercice 4 Le marchand de primeurs

Dans cet exercice on veut simuler la gestion d'un magasin de fruits et légumes en Ocaml, avec des types adaptés.

Un produit sera représenté par sa catégorie, son nom, la quantité présente en rayon (en kg) et son prix.

Le magasin vend 4 catégories de produits : les fruits, les légumes, les jus de fruits et les fruits secs. Les noms des produits en revanche sont trop nombreux pour être énumérés, on considérera que ce sont des chaines de caractères (type `string`).

1. Définir un type énumération `categorie` pour définir la categorie d'un produit.

```
type categorie = Fruit | Legume | Jus | Noix;;
```

2. Définir un type `produit` qui permet de représenter un produit du magasin.

```
type produit = {cat : categorie; nom : string; quant : float; prix : float};;
```

3. Écrire une fonction `est_en_stock : produit -> bool` qui indique si un produit est présent en rayon (il faut qu'il en reste strictement plus que 0 kg).

```
let est_en_stock p = p.quant > 0.;;
```

On représentera le magasin par la liste des références qu'il vend. On définit l'alias `type magasin = produit list`.

4. Écrire une fonction `mise_en_rayon : string -> float -> magasin -> magasin` qui étant donné un nom de produit et une quantité, augmente le stock de ce produit en magasin de la quantité donnée. On supposera que le produit était déjà vendu par le magasin.

```
let rec mise_en_rayon n pds mag = match mag with
 |h::q when h.nom = n -> {cat = h.cat; nom = h.nom; quant = h.quant+.pds; prix = h.prix}::q
 |h::q -> mise_en_rayon n pds q;;
```

5. Écrire une fonction `est_nouvelle_reference : string -> magasin -> bool` qui détermine si le magasin vend déjà un produit du nom donné.

```
let rec est_nouvelle_reference n mag = match mag with
 |[] -> false
 |h::q when h.nom = n -> true
 |h::q -> est_nouvelle_reference n q;;
```

6. Écrire une fonction `change_catalogue : produit -> magasin -> magasin` qui met à jour le magain en ajoutant un nouveau produit.

```
let rec change_catalogue p mag = p::mag;;
```

7. Écrire une fonction `livraison : produit list -> magasin -> magasin` qui effectue la mise à jour des stocks après une livraison.

```
let rec livraison commande mag = match commande with
  |[] -> []
  |h::q when est_nouvelle_reference h.nom mag -> change_catalogue h (livraison q mag)
  |h::q -> livraison q (mise_en_rayon h.nom h.quant mag);;
```

On représentera le panier d'un acheteur par une liste de couples `(n, kg)` indiquant que le client souhaite acheter le produit de nom `n`, en quantité `kg`.

8. Écrire une fonction `achats : (string * float) list -> magasin -> (magasin * float)` qui renvoie le nouveau stock du magasin en fonction des achats réalisés et le total que le client paye en caisse.

```
let rec achats panier mag =
  let rec modif n pds l = match l with
    |[] -> ([],0.)
    |h::q when h.nom = n -> let newh = {cat = h.cat; nom = h.nom; quant = max (h.quant-.pds) 0.; prix = h
                           in (newh::q,h.prix*.(min h.quant pds))
    |h::q -> modif n pds q
  in
  match panier with
  |[] -> ([],0.)
  |(n,pds)::q -> let mag1, p1 = modif n pds mag in let mag2, p2 = achats q mag1 in (mag2, p2+.p1);;
```

9. Écrire une fonction `commande : magasin -> produit list` qui étant donné un magasin renvoie la commande à effectuer pour acheter les produits qui manquent.

```
let rec commande mag = match mag with
  |[] -> []
  |h::q when not (est_en_stock h) ->
          let newpds = match h.cat with
                |Fruit -> 20.
                |Legume -> 30.
                |Jus -> 10.
                |Noix -> 3. in {cat = h.cat; nom = h.nom; quant = newpds; prix = h.prix}::(commande q)
  |h::q -> commande q;;
```