DEUG 1^{ere} année
MIAS 1 et MASS 1 – Programmation fonctionnelle

TD n° 7 – Corrigé

Exercice 1 :
Complexité en temps. Le paramètre d'induction de la fonction `ajoutefin` est la liste `l`. Le temps nécessaire pour calculer le résultat de cette fonction ne dépend que de la liste `l`, on évalue donc la complexité en temps en fonction de la longueur de cette liste.

Notons :

- n la longueur de la liste `l`;
- C le temps (maximum) consommé pour évaluer le corps de la fonction, c'est à dire le temps consommé pour évaluer l'expression conditionnelle sans compter le temps nécessaire pour évaluer l'éventuel appel récursif, ce temps ne dépend pas de la liste et est constant;
- $T(n)$ le temps consommé pour calculer le résultat de `ajoutefin` pour une liste de longueur n.

Si la liste est vide, le calcul de `ajoutefin` nécessite uniquement d'évaluer le corps de la fonction. Si la liste n'est pas vide, elle est de longueur n, et le temps consommé est celui nécessaire pour évaluer le corps de la fonction auquel on ajoute le temps nécessaire au calcul de `ajoutefin` pour une liste de longueur $n-1$ (l'appel récursif). Le temps consommé est donc donné par les équations :

$$\begin{cases} T(0) &= C \\ T(n) &= C + T(n-1) \quad \text{, pour } n \geq 1 \end{cases}$$

$T(n)$ est une suite arithmétique de premier terme C et de raison C, on a donc $T(n) = (n+1)C = O(n)$. Le temps consommé pour calculer `ajoutefin` est donc proportionnel à la longeur de la suite.

Complexité en espace. Comme pour la complexité en temps, la complexité en espace est fonction de la longueur n de la liste `l`. On doit calculer l'espace mémoire maximum nécessaire pour effectuer le calcul de `ajoutefin`. Pour cela, on va calculer le nombre de symboles de la plus longue expression que l'interprète doit conserver en mémoire pour effectuer le calcul.

Notons : $E(n)$ ce nombre.

Si la liste est vide, le résultat de la fonction est donné par l'évaluation de l'expression `(cons o l)` qui contient 3 symboles. Si la liste n'est pas vide, l'interprète doit évaluer l'expression `(cons (car l) (ajoutefin o (cdr l)))` dont le nombre de symboles est 3 augmenté du nombre de symboles de la plus longue expression que l'interprète doit évaluer pour calculer `(ajoutefin o (cdr l))`, c'est à dire $E(n-1)$. La complexité en espace est donc donnée par les équations :

$$\begin{cases} E(0) &= 3 \\ E(n) &= 3 + E(n-1) \quad \text{, pour } n \geq 1 \end{cases}$$

C'est encore une suite arithmétique dont le terme générale vaut $E(n) = 3(n+1) = O(n)$. L'espace mémoire maximum nécessaire pour calculer `ajoutefin` est aussi proportionnel à la longueur de la liste.

Exercice 2 :
Complexité en temps de `sauf`. Le temps consommé ne dépend que de la longueur de la liste.

Notons :

- n la longueur de la liste `l`;
- C_s le temps (maximum) consommé pour évaluer le corps de la fonction sans compter le temps nécessaire pour évaluer l'éventuel appel récursif, ce temps est constant;
- $T_s(n)$ le temps consommé pour calculer le résultat de `sauf` pour une liste de longueur n.

Si la liste est vide, le temps consommé est égal à C_s. Si la liste n'est pas vide, sa longueur vaut n, le temps consommé est C_s augmenté du temps nécessaire pour évaluer l'appel récursif. Deux cas peuvent conduire à un appel récursif et dans les deux cas l'appel se fait sur une liste de longeur $n-1$ et consomme donc un temps $T_s(n-1)$. On a donc :

$$\begin{cases} T_s(0) &= C_s \\ T_s(n) &= C_s + T_s(n-1) \quad \text{, pour } n \geq 1 \end{cases}$$

Soit encore $T_s(n) = (n+1)C_s = O(n)$. La complexité en temps de la fonction `sauf` est linéaire.

Complexité en temps de `enleve`. L'appel récursif de cette fonction se fait sur la liste `l1` et donc la complex
temps dépend de la longueur cette liste. Mais ce n'est pas tout car cette fonction fait appel à la fonction `sauf`
aussi consommer un temps qui est fonction de la longueur de la liste `l2` au moment où l'appel à lieu. La comp
en temps de `enleve` est donc fonction des longueurs de `l1` et `l2`.

Notons :

- n_1 la longueur de la liste `l1`;
- n_2 la longueur de la liste `l2`;
- C_e le temps (maximum) consommé pour évaluer le corps de la fonction `enleve` sans compter le temps néc
pour évaluer l'éventuel appel récursif, ce temps est constant;
- $T_e(n1, n2)$ le temps consommé pour calculer le résultat de `enleve`.

Si la liste `l1` est vide, aucun appel récursif n'est nécessaire, le temps consommé ne dépend pas de la longueur d
est égal à C_e.
Si `l1` est de longueur n_1, le temps consommé est C_e, auquel il faut ajouter le temps consommé par un app
fonction `enleve` sur la liste `(cdr l1)` de longueur $n_1 - 1$, ainsi que le temps consommé par l'appel à la fonctio
sur la liste `l2`.
Seulement, au fur et à mesure des appels, la longueur de la liste `l2` peut diminuer (si on enleve effectiveme
éléments). Or il faudrait connaître la longueur de `l2` à chaque appel à `sauf` pour savoir exactement quel tem
consommé. Comme cela dépend des éléments que contiennent les listes `l1` et `l2` et qu'on ne peut le savoir a pri
évalue donc le temps consommé dans le pire des cas, c'est à dire lorsqu'aucun des éléments de `l1` n'est présen
`l2`; dans ce cas la liste `l2` reste de taille constante et le temps consommé par chaque appel à `sauf` vaut $(n_2 +$
On obtient ainsi une borne supérieure du temps consommé $T_e(n_1, n_2)$ qui est donné par les équations :

$$\begin{cases} T_e(0, n_2) &= C_e \\ T_e(n_1, n_2) &= C_e + T_e(n_1 - 1, n_2) + T_s(n_2) = C_e + T_e(n_1 - 1, n_2) + (n_2 + 1)C_e \quad \text{, pour } n_1 \geq 1 \end{cases}$$

Pour résoudre ces équations, on peut écrire tous les termes et en faire l'addition :

$$\begin{array}{lcllll} T_e(n_1, n_2) & = & C_e & + & T_e(n_1 - 1, n_2) & + & (n_2 + 1)C_e \\ T_e(n_1 - 1, n_2) & = & C_e & + & T_e(n_1 - 2, n_2) & + & (n_2 + 1)C_e \\ T_e(n_1 - 2, n_2) & = & C_e & + & T_e(n_1 - 3, n_2) & + & (n_2 + 1)C_e \\ \dots & & & & & & \\ T_e(2, n_2) & = & C_e & + & T_e(1, n_2) & + & (n_2 + 1)C_e \\ T_e(1, n_2) & = & C_e & + & T_e(0, n_2) & + & (n_2 + 1)C_e \\ T_e(0, n_2) & = & C_e & & & & \\ \hline T_e(n_1, n_2) & = & (n_1 + 1)C_e & + & (n_1 + 1)(n_2 + 1)C_s & & \end{array}$$

La complexité en temps de `enleve` est donc fonction des longueurs des listes `l1` et `l2` est vaut au ma
$T_e(n_1, n_2) = (n_1 + 1)C_e + (n_1 + 1)(n_2 + 1)C_s = O(n_1 n_2)$. Le temps maximum consommé est proportion
produit des longueurs des deux listes.

Exercice 3:

```
(define insere                                          (define tri
  (lambda (e l)                                           (lambda (l)
    (cond ((null? l) (cons e l))                            (if (null? l)
          ((<= e (car l)) (cons e l))                           l
          (else (cons (car l) (insere e (cdr l)))))))           (insere (car l) (tri (cdr l
```

1) Complexité en temps de `insere`. Le temps consommé par `insere` dépend de la longueur de la liste `l`.

Notons :

- n la longueur de la liste `l`;
- C_i le temps (maximum) consommé pour évaluer le corps de la fonction sans compter le temps nécessair
évaluer l'éventuel appel récursif, ce temps est constant;
- $T_i(n)$ le temps consommé pour calculer le résultat de `insere` pour une liste de longueur n.

Si la liste est vide, on a $T_i(0) = C_i$.
Si la liste est de longueur n, deux cas sont possibles :
– soit l'élément e est plus petit que le premier élément de la liste, dans ce cas aucun appel récursif n'est nécessaire et le temps consommé est $T_i(n) = C_i$;
– soit lélément e est plus grand que le premier élément de la liste, dans ce cas le temps consommé est $T_i(n) = C_i + T_i(n-1)$ à cause de l'appel à `insere` sur le `(cdr l)`.

Comme on ne connaît pas a priori la valeur des éléments de la liste, on évalue le temps consommé dans le pire des cas. On a donc :

$$\begin{cases} T_i(0) &= C_i \\ T_i(n) &= C_i + T_i(n-1) \quad \text{, pour } n \geq 1 \end{cases}$$

Soit $T_i(n) = (n+1)C_i = O(n)$.

2) Complexité en temps de `tri`.

Notons :

- n la longueur de la liste `l`;
- C_t le temps (maximum) consommé pour évaluer le corps de la fonction sans compter le temps nécessaire pour évaluer l'éventuel appel récursif, ce temps est constant;
- $T_t(n)$ le temps consommé pour trier une liste de longueur n.

Si la liste est vide, le temps consommé est $T_t(0) = C_t$.
Si la liste est de longueur n il faut compter le temps consommé par le tri de `(cdr l)`, soit $T_t(n-1)$, ainsi que le temps maximum consommé par l'appel à la fonction `insere` sur la liste résultat de `(tri (cdr l))` qui est de longeur $n-1$.
On a donc :

$$\begin{cases} T_t(0) &= C_t \\ T_t(n) &= C_t + T_t(n-1) + T_i(n-1) = C_t + T_t(n-1) + nC_i \quad \text{, pour } n \geq 1 \end{cases}$$

Pour résoudre ces équations, on procède par sommation :

$$\begin{array}{lcllcll} T_t(n) & = & C_t & + & T_t(n-1) & + & nC_i \\ T_t(n-1) & = & C_t & + & T_t(n-2) & + & (n-1)C_i \\ T_t(n-2) & = & C_t & + & T_t(n-3) & + & (n-2)C_i \\ T_t(2) & = & C_t & + & T_t(1) & + & 2C_i \\ T_t(1) & = & C_t & + & T_t(0) & + & 1C_i \\ T_t(0) & = & C_t & & & & \\ \hline T_t(n) & = & (n+1)C_t + \sum_{k=1}^{n} kC_i & & & & \end{array}$$

Soit finalement $T_t(n) = (n+1)C_t + \frac{n(n+1)}{2}C_i = O(n^2)$.

3) Complexité dans le meilleurs des cas. Si la liste est déjà triée en ordre croissant, chaque appel à la fonction `insere` se fait avec un élément qui est plus petit que tous ceux de la liste, dans ce cas le temps consommé par `insere`, quelque soit la longeur de la liste, est constant et vaut $T_i(n) = C_i$. Le temps consommé par `tri` est alors donné par les équations :

$$\begin{cases} T_t(0) &= C_t \\ T_t(n) &= C_t + T_t(n-1) + C_i \quad \text{, pour } n \geq 1 \end{cases}$$

dont la solution est $T(n) = C_t + n(C_t + C_i) = O(n)$.