TP numéro 3 : tris, listes de listes

1. Tri par insertion

Nous allons trier une liste en utilisant le tri par insertion. Le principe de ce tri consiste à trier récursivement le *cdr* de la liste, puis à insérer le *car* au bon endroit dans la liste.

• Écrire la fonction `insere` qui insère au bon endroit un nombre dans une liste de nombres triée.

```
(insere 4 '(1 2 5 7)) → '(1 2 4 5 7)
```

• Écrire la fonction `tri-insertion` qui trie une liste de nombres.

```
(tri-insertion '(7 4 9 1)) → (1 4 7 9)
```

2. Listes de listes : la suite de Conway

Devinette : comment se construit la suite de Conway ?

0, 10, 1110, 3110, 132110, 1113122110, 311311222110, ...

• On veut écrire une fonction qui calcule un terme de la suite de Conway à partir du précédent. Si vous n'avez pas trouvé la réponse à la devinette, demandez à votre enseignant ou à wikipedia.

```
(conway '(1 1 1 0)) -> (3 1 1 0)
```

Vous pouvez écrire cette fonction en remontant (en utilisant le résultat de l'appel récursif sur le reste de la liste) ou en descendant (en utilisant un compteur du nombre de répétitions de l'élément courant).

• Écrire une fonction qui calcule les n premiers termes de la suite de Conway en fonction du premier terme.

```
(conwayn 4 '(0)) -> ((0) (1 0) (1 1 1 0) (3 1 1 0))
```

3. D'autres listes de listes

• Définir une fonction `ajoute` qui insère un élément en tête de la sous-liste dont l'indice est passé en paramètre (l'indice est forcément inférieur ou égal à la taille de la liste).

```
(ajoute 'a '((e r) (r y b) (t e)) 2) → ((e r) (a r y b) (t e))
(ajoute 'a '((e r) (r y b) (t e)) 0) → ((a) (e r) (r y b) (t e))
```

• Définir une fonction `(sp n x y)` qui, étant donnés deux nombres x et y, calcule les n premiers termes de la suite $x_n = x_{n-1} + y_{n-1}$ et $y_n = x_{n-1} * y_{n-1}$.

```
(sp 4 5 2) → ((5 2) (7 10) (17 70) (87 1190))
```

4. Listes ordonnées

• Définir une fonction qui, étant donné un nombre n et une liste l de nombres triée en ordre croissant, retourne la liste des éléments de l diviseurs de n. *Attention à ne pas parcourir la liste en totalité quand ce n'est pas nécessaire.*

```
(diviseurs '(1 2 3 4 5 8 12 13 15 17) 12) → (1 2 3 4 12)
```