TP numéro 3

Tri par sélection du maximum

On veut définir une fonction qui trie en ordre croissant une liste de nombres en sélectionnant le maximum de cette liste puis en triant récursivement les autres nombres de la liste.

• Définir une fonction `maximum` qui calcule le maximum d'une liste de nombres non vide.

```
(maximum '(1 8 5 2 6)) → 8
```

• Définir une fonction `supprime` qui supprime un élément d'une liste (on se contentera de supprimer la première occurrence de l'élément).

```
(supprime 5 '(2 8 5 6 4)) → (2 8 6 4)
```

• Définir la fonction `tri-max` qui utilise les deux précédentes pour trier une liste de nombres par sélection du maximum.

```
(tri-max '(8 2 4 3 1 5 9)) → (1 2 3 4 5 8 9)
```

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)
```

• Définir une fonction qui, étant donné un nombre n et une liste l de nombres triée en ordre croissant, retourne une liste de deux listes : la liste des éléments de l inférieurs à n, et celle des éléments de l supérieurs ou égaux à n. Attention à ne pas parcourir la liste en totalité quand ce n'est pas nécessaire.

```
(separe '(1 3 4 6 8 9 12) 7) → ((1 3 4 6) (8 9 12))
```

Listes de listes

• Définir une fonction `ajoute` qui insère un élément en tête de la sous-liste dont l'indice est passé en paramètre.

```
(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))
```

Calculs en descendant ou en remontant

Nous allons voir deux manières de programmer la fonction qui calcule le nombre d'occurrences d'un élément donné dans une liste donnée.

1. Définissez une version récursive de cette fonction de la manière habituelle, en utilisant le résultat de l'appel récursif sur le reste de la liste.

```
(occurrence 'a '(a b c a e r d a t)) → 3
```

2. Envisageons maintenant une version qui, bien qu'étant récursive, s'inspire de la programmation itérative, en utilisant un compteur. On définit pour cela une fonction `occ` qui retourne le même résultat, mais prend un argument supplémentaire : un entier qui sert à compter le nombre d'occurrences de l'élément dans la liste.

La fonction `occurrence` s'écrira alors :

```
(define occurrence ;  -> entier
  (lambda (x l) ; x element, l liste
    (occ x l 0)))
```

Effectuer une trace de l'exécution des deux versions de la fonction sur un exemple simple.