

**Exercice 1.** \* Écrire une fonction qui divise une liste en deux, en mettant un élément sur deux dans une liste, un élément sur deux dans une autre.

Exemple :

```
# divise_en_deux [1;2;3;4;5;6;7;8;9];;  
- : int list * int list = ([1;3;5;7;9], [2;4;6;8])
```

Je me moque de savoir dans quel ordre sont vos deux listes à l'arrivée. Donc

```
# divise_en_deux [1;2;3;4;5;6;7;8;9];;  
- : int list * int list = ([9;7;5;3;1], [2;4;6;8])
```

me convient aussi bien.

Quelle est la complexité algorithmique de cette fonction, en nombre de filtrages par motifs ?

**Exercice 2.** \* Écrire une fonction qui fusionne de deux listes triées (de taille peut-être distinctes) en une seule liste triée.

Exemple :

```
# fusionne_triees [1;3;5;7;9;11;13;17], [2;4;6;8]  
- : int list = [1;2;3;4;5;6;7;8;11;13;17]
```

Quelle est la complexité algorithmique de cette fonction, en nombre de ::.

**Exercice 3.** \* Écrire une fonction qui divise une liste en deux, en mettant tous les éléments plus petits que  $p$  dans une liste, tous les éléments plus grands que  $p$  dans une autre.

Exemple :

```
# compare_a 3 [14;1;2;3;4;5;6;7;8;9];;  
- : int list * int list = ([1;2;3], [14;4;5;6;7;8;9])
```

Même remarque que au-dessus sur l'ordre des éléments.