Exercice 2.3 (Concaténation)

Écrire une fonction qui concatène deux listes (l'équivalent de l'opérateur `@`).

Exemple d'application :

```
# append [1;2;3;4] [5;6];;
- : int list = [1; 2; 3; 4; 5; 6]
```

3 Listes et ordres

Exercice 3.1 (Croissance ?)

Écrire une fonction qui teste si une liste est triée en ordre croissant.

Exercice 3.2 (Recherche)

Écrire une fonction qui recherche si un élément est présent dans une liste triée (en ordre croissant).

Exercice 3.3 (Association – *C1 - nov. 2017*)

Écrire la fonction `assoc k list` où `list` est une liste de couples (*key*, *value*) triés par clés (*key*) croissantes. Les clés sont des entiers naturels non nuls. Elle retourne la valeur (*value*) associée à la clé (*key*) `k`. Si `k` n'est pas valide ou si aucun couple n'a pour clé `k`, elle déclenche une exception.

Exemples d'utilisation :

```
# assoc 5 [(1, "one"); (2, "two"); (3, "three"); (5, "five"); (8, "eight")];;
- : string = "five"

# assoc 4 [(1, "one"); (2, "two"); (3, "three"); (5, "five"); (8, "eight")];;
Exception: Failure "not found".

# assoc (-1) [(1, "one"); (2, "two"); (3, "three"); (5, "five"); (8, "eight")];;
Exception: Invalid_argument "k not a natural".
```

Exercice 3.4 (Suppression)

Écrire une fonction qui supprime d'une liste l triée (en ordre croissant) la première occurrence d'un élément x (s'il est présent).

Exercice 3.5 (Insertion)

Écrire une fonction qui ajoute un élément à sa place dans une liste triée en ordre croissant.

Exercice 3.6 (Inverse)

Écrire une fonction qui inverse une liste :

1. en utilisant l'opérateur `@` ;
2. sans utiliser l'opérateur `@`.

Que pensez-vous de la complexité de ces deux fonctions ?