

**Exercice 1** :

1. On reprend la fonction `retire` écrite au TP précédent.

```

1 let rec retire l x =
2   match l with
3   | [] -> []
4   | y::q -> if y = x then retire q x else y::retire q x

```

- (a) Soit  $C(n, x)$  la complexité dans le pire cas de cette fonction pour une liste  $l$  de taille  $n$  et une valeur  $x$  (de type numérique). Établir une relation de récurrence (on ne demande pas les constantes exactes) sur  $C(n, x)$ .
- (b) En déduire que  $C(n, x)$  ne dépend pas de  $x$  et est  $\mathcal{O}(n)$ .
2. On rappelle que `nth l i` est de complexité dans le pire cas linéaire en  $\min(n, i)$ , où  $n$  est la taille de  $l$ .

(a) Montrer que la fonction suivante (volontairement mal écrite, on peut faire plus rapide) a une complexité dans le pire cas quadratique en  $\min(n, i)$ , avec les mêmes notations.

```

1 let rec mystere l i =
2   if i == -1 then []
3   else (nth l i)::(mystere l (i - 1))

```

(b) Que calcule `mystere l (length l)` ?

**Exercice 2** :

1. (a) Écrire une fonction récursive `concat : 'a list -> 'a list -> 'a list` qui prend en argument deux listes d'éléments de même type et renvoie une liste composée des éléments de la première suivi des éléments de la deuxième.

Exemple : `concat [1; 42; 3] [3; 17; -2; 5] = [1; 42; 3; 3; 17; -2; 5]`.

À l'avenir, plutôt que réécrire cette fonction vous pouvez utiliser l'opérateur `@`.

- (b) Comment sont représentées en mémoire les listes argument de `concat` et la liste résultat ?
- (c) Quelle est l'ordre de grandeur de la complexité de `concat` en fonction de  $n_1$  et  $n_2$  les tailles des listes en argument ?
2. On souhaite écrire une fonction pour calculer le renversé d'une liste (à l'avenir on pourra utiliser `List.rev` pour cela).

(a) Pourquoi ne peut-on pas simplement écrire une fonction récursive de complexité linéaire en la taille de la liste `rev : 'a list -> 'a list` qui fait ce qu'on veut ?

(b) Écrire une fonction récursive `rev_aux : 'a list -> 'a list -> 'a list` telle que `rev l1 l2` calcule le renversé de `l1` suivi de `l2`.

Exemple : `rev_aux [1; 2; 42] [12; 18] = [42; 2; 1; 12; 18]`.

(c) En utilisant la fonction ci-dessus, écrire `rev`.

3. Écrire une fonction récursive `for_all : ('a -> bool) -> 'a list -> bool` telle que `for_all p l`, avec  $l = [x_0; x_1; \dots; x_{n-1}]$  est vrai si et seulement si  $\forall i \in \llbracket 0; n-1 \rrbracket, p(i)$ .

Exemples :

- `for_all (fun x -> x > 0) [3; 17; -2; 5] = false.`
- `for_all (fun x -> x > 0) [3; 17; 5] = true`

À l'avenir, plutôt que réécrire cette fonction vous pouvez utiliser la fonction `List.for_all`.