TD - Preuves d'algorithmes
Terminaison - Correction.

Exercice 1 (Échauffement - Somme).
On donne le code suivant :

```
let sum t =
  let s = ref 0 in
  let n = Array.length in
  for i = 0 to n-1 do
    s := !s + t.(i)
  done;
  !s;;
```

1. Que fait ce code ? Donner le prototype OCaml de cette fonction.
2. Prouver la terminaison de cet algorithme.
3. Prouver la correction de cet algorithme.

Exercice 2 (Tri par insertion).

1. Écrire une fonction récursive OCaml nommée `insere`, qui insère une nouvelle valeur dans une liste d'entiers **déjà triée** par ordre croissant, en respectant cet ordre.
2. Prouver la terminaison de cette fonction.
3. Prouver la correction de cette fonction.
4. Écrire une fonction récursive OCaml `tri_insertion` qui implémente le tri par insertion.
5. Prouver la terminaison de cette fonction.
6. Prouver la correction de cette fonction.

Exercice 3 (Multiplication russe).

1. Cours : réécrire l'algorithme de multiplication russe sous forme récursive **terminale** en OCaml.
2. Prouver la terminaison de cet algorithme.
3. Démontrer que :
$$\forall x \in \mathbb{R}, \forall q \in \mathbb{N}^*, \left\lfloor \frac{\lfloor x \rfloor}{q} \right\rfloor = \left\lfloor \frac{x}{q} \right\rfloor$$
4. En déduire une formule explicite pour le variant choisi.
5. Combien d'appels récursifs sont nécessaires pour que l'algorithme termine ?
6. Qu'en déduire sur la complexité temporelle de cet algorithme ?

Entraînement à la maison

Exercice 4 (Occurrences d'une valeur dans un tableau).
On considère un algorithme **séquentiel** nommé `nombre_occurrences` qui renvoie le nombre d'occurrences d'une valeur v dans un tableau d'entiers.

1. Écrire une fonction implémentant cet algorithme en C.
2. Prouver la terminaison de l'algorithme.
3. Prouver la correction de l'algorithme.

Exercice 5 (Preuves de terminaison).
Montrez que les fonctions suivantes terminent. **Les arguments donnés en entrée de ces fonctions sont tous des entiers.**

```
let rec g a =
  if a < 0 then 1
  else if a mod 2 = 0 then g (a+1)
  else g (a-3)
```

```
let rec f a b =
  if a <= 0 || b <=0 then 1
  else if a mod 2 = 0 then f (a/3) (2*b)
  else f (3*a) (b/5)
```

En cas de panne d'inspiration, vous pouvez coder et introduire des affichages dans ces fonctions pour bien comprendre leur fonctionnement sur des exemples et ainsi vous persuader de leur terminaison.