Exercice 1. Terminaison
Montrer que les fonctions suivantes terminent, pour des arguments 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)
```

Exercice 2. Invariant de boucle simple
En utilisant un invariant de boucle, prouver que la fonction suivante renvoie bien la somme des éléments d'un tableau :

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

On rappelle qu'un invariant de boucle est une propriété qui reste vraie à chaque itération de la boucle et qui permet de montrer que la fonction renvoie le bon résultat (ici, la somme des éléments de `t`).
On prouve cet invariant de boucle par récurrence sur le nombre d'itérations dans la boucle.

Exercice 3. Tranche maximum (algorithme de Kadane)
Soit `t` un tableau d'entiers. Une **somme consécutive** (ou tranche) dans `t` est de la forme $\displaystyle\sum_{k=i}^{j}$ `t.(k)` (où i et j sont des indices de `t`). On note s la valeur maximum d'une somme consécutive.

1. Écrire une fonction `tranche_max` prenant `t` en argument et renvoyant `s`, en complexité quadratique en la taille de `t`.

Si j est un indice de `t`, on note s_j la plus grande somme consécutive finissant en j. Dit autrement :

$$s_j = \max_{0 \leq i \leq j} \sum_{k=i}^{j} t.(k)$$

2. Calculer tous les s_j, si `t = [|1; -4; 1; 5; -7; 0|]`
3. Si $j > 0$, montrer que :
$$s_j = \max(s_{j-1} + t.(j),\ t.(j))$$
4. Comment peut-on exprimer s en fonction de s_j ?
5. En déduire une fonction `tranche_max` prenant `t` en argument et renvoyant `s`, en complexité linéaire en la taille de `t`.
6. Donner un invariant de boucle permettant de prouver que `tranche_max t` est correct.
7. Modifier votre fonction précédente pour obtenir les indices de début et fin de s.

Exercice 4. Tri par insertion

1. Écrire une fonction `insere` telle que, si `l` est une liste triée et `e` un élément, `insere l e` renvoie une liste triée contenant `e` et les éléments de `l`.
2. En déduire un algorithme de tri, en utilisant plusieurs fois `insere`. Prouver que ce tri est correct.

3. Quelle est la complexité de ce tri ? Pourrait-on l'améliorer en utilisant une recherche par dichotomie pour `insere` ?
4. Réécrire `tri_insertion` avec un tableau au lieu d'une liste et en utilisant une recherche par dichotomie pour l'insertion. Que peut-on dire de sa complexité ? Et de son nombre de comparaisons ?