Le tri fusion sur les listes.

À partir de deux listes triées, on peut fusionner les deux listes pour former une nouvelle liste triée.

```
#let rec fusion = fun
  l1 [] -> l1
 | [] l2 -> l2
 | (x::xs as l1) (y::ys as l2) -> if x < y then x::(fusion xs l2)
                                           else y::(fusion l1 ys);;
fusion : 'a list -> 'a list -> 'a list = <fun>
#fusion [1;2;5;9;44] [4;5;7;8;9;12;56];;
- : int list = [1; 2; 4; 5; 5; 7; 8; 9; 9; 12; 44; 56]
```

On peut imaginer une fonction de tri_fusion qui à partir d'une liste l commence par séparer la liste en deux listes de longueurs comparables puis applique le «tri fusion» à chacune des sous-listes et enfin fusionne les deux sous-listes triées.

```
#let rec separe = function
[] -> ([],[])
| [a] -> ([a],[])
| (x::y::xs) -> let (l1,l2) = separe xs in
                    (x::l1,y::l2);;
separe : 'a list -> 'a list * 'a list = <fun>
#separe [1;2;3;4;5;6;7;8;9;10];;
- : int list * int list = [1; 3; 5; 7; 9], [2; 4; 6; 8; 10]
```

```
#let rec tri_fusion = function
 [] -> []
| [x] -> [x]
| l -> let (l1,l2) = separe l in
          fusion (tri_fusion l1) (tri_fusion l2);;
tri_fusion : 'a list -> 'a list = <fun>
#tri_fusion [];;
- : '_a list = []
#tri_fusion [1];;
- : int list = [1]
#tri_fusion [1;-8;6;4;12;8;5;6;4;7;9;12];;
- : int list = [-8; 1; 4; 4; 5; 6; 6; 7; 8; 9; 12; 12]
```

Le tri par insertion sur les listes.

Le principe est le suivant : si on dispose d'une liste `x::xs`, on commence par trier la liste `xs` puis on insère l'élément `x` à sa place dans la liste `xs` triée.
Commençons par écrire une fonction d'insertion d'un élément `x` dans une liste triée `l` :

```
#let rec insertion x = function
  [] -> [x]
| (y::ys as l) when x <= y -> x::l
| y::ys -> y::(insertion x ys);;
insertion : 'a -> 'a list -> 'a list = <fun>
#let l = [-8; 1; 4; 4; 5; 6; 6; 7; 8; 9; 12; 12];;
l : int list = [-8; 1; 4; 4; 5; 6; 6; 7; 8; 9; 12; 12]
#let l = insertion 0 l;;
l : int list = [-8; 0; 1; 4; 4; 5; 6; 6; 7; 8; 9; 12; 12]
#let l = insertion 40 l;;
l : int list = [-8; 0; 1; 4; 4; 5; 6; 6; 7; 8; 9; 12; 12; 40]
#let l = insertion (-10) l;;
l : int list = [-10; -8; 0; 1; 4; 4; 5; 6; 6; 7; 8; 9; 12; 12; 40]
```

À partir de là, on peut facilement imaginer une fonction de tri par insertion :

```
#let rec tri_insertion = function
 [] -> []
| [x] -> [x]
| x::xs -> insertion x (tri_insertion xs);;
tri_insertion : 'a list -> 'a list = <fun>
#tri_insertion [1;-8;6;4;12;8;5;6;4;7;9;12];;
- : int list = [-8; 1; 4; 4; 5; 6; 6; 7; 8; 9; 12; 12]
```

Le tri par bulles sur les listes.

Le tri par bulles ne peut pas s'appliquer de façon naturelle aux listes. En effet dans le principe, on fait «remonter les bulles» du début à la fin du tableau puis du début à l'avant dernière position,..., puis du début à la $i^{ème}$ position,.... Or dans un tableau, un élément n'est pas repérable par sa position. On peut malgré tout adapter cet algorithme de tri au prix d'une dégradation des performance, en faisant remonter les bulles du début jusqu'à la fin de la liste autant de fois qu'il y a d'éléments dans la liste, ce qui nous donne les fonctions suivantes :

```
#let rec bulle = function
 [] -> []
| [x] -> [x]
| x::y::xs -> if x > y then y::(bulle (x::xs))
                       else x::(bulle (y::xs));;
bulle : 'a list -> 'a list = <fun>
```

```
#let tri_bulle l =
let liste = ref l in
for i = 1 to list_length l do
    liste := bulle !liste
done;
!liste;;
tri_bulle : 'a list -> 'a list = <fun>
#tri_bulle [9;8;7;6;5;4;3;2;1;0];;
- : int list = [0; 1; 2; 3; 4; 5; 6; 7; 8; 9]
```

exercices : On suppose que l'on représente des ensembles finis par des listes triées. Écrire les fonctions suivantes :

- fonction qui retourne la réunion de deux ensembles,
- fonction qui retourne l'intersection de deux ensembles,
- fonction qui retourne la différence symétrique de deux ensembles,