Algorithmique
Correction Contrôle n° 1
(Version profs)

INFO-SUP S1# – EPITA

Solution 1 **(Types Abstraits : Vecteur (erreurs et extension) – *6 points*)**

1. En fait il y a deux sortes de problèmes :
   — un de complétude (manque d'axiomes) avec l'absence d'axiome appliquant `estinitialisé` à `vect`.
   — un de consistance (ambiguïté entre plusieurs axiomes) avec l'absence de i≠j sur l'un des axiomes appliquant `estinitialisé` à `modifième`.

   La déclaration corrigée du type `vecteur` devrait être :

   TYPES
   vecteur

   UTILISE
   entier, élément, booléen

   OPÉRATIONS

vect	:	entier × entier → vecteur
modifième	:	vecteur × entier × élément → vecteur
ième	:	vecteur × entier → élément
estinitialisé	:	vecteur × entier → booléen
borneinf	:	vecteur → entier
bornesup	:	vecteur → entier

   PRÉCONDITIONS
   ième(v,i) est-défini-ssi borneinf(v) ⩽ i ⩽ bornesup(v) & estinitialisé(v,i)=vrai

   AXIOMES
   borneinf(v) ⩽ i ⩽ bornesup(v) ⇒ ième(modifième(v,i,e),i) = e
   borneinf(v) ⩽ i ⩽ bornesup(v) & borneinf(v) ⩽ j ⩽ bornesup(v) & i≠j
   ⇒ ième(modifième(v,i,e),j) = ième(v,j)
   estinitialisé(vect(i,j),k)=Faux
   borneinf(v) ⩽ i ⩽ bornesup(v) ⇒ estinitialisé(modifième(v,i,e),i)=vrai
   borneinf(v) ⩽ i ⩽ bornesup(v) & borneinf(v) ⩽ j ⩽ bornesup(v) & i≠j
   ⇒ estinitialisé(modifième(v,i,e),j)=estinitialisé(v, j)
   borneinf(vect(i,j))=i
   borneinf(v) ⩽ i ⩽ bornesup(v) ⇒ borneinf(modifième(v,i,e))=borneinf(v)
   bornesup(vect(i,j))=j
   borneinf(v) ⩽ i ⩽ bornesup(v) ⇒ bornesup(modifième(v,i,e))=bornesup(v)

   AVEC

vecteur	v
entier	i, j, k
element	e

2. Extension au type *vecteur* :

(a) Il n'y a pas de préconditions. C'est une opération interne définie partout sur les bornes du vecteur. Les limites seront précisées si nécessaires dans les postulat.

(b) Les axiomes sont les suivants :

AXIOMES

borneinf(v) ⩽ i ⩽ bornesup(v) ⇒ estinitialisé(réinitialise(v,i),i)=faux
borneinf(v) ⩽ i ⩽ bornesup(v) & borneinf(v) ⩽ j ⩽ bornesup(v) & i≠j
⇒ estinitialisé(réinitialise(v,i),j) = estinitialisé(v,j)
borneinf(v) ⩽ i ⩽ bornesup(v) & borneinf(v) ⩽ j ⩽ bornesup(v) & i≠j
⇒ ième(réinitialise(v,i),j) = ième(v,j)
borneinf(réinitialise(v,i))=borneinf(v)
bornesup(réinitialise(v,i))=bornesup(v)

AVEC

vecteur v
Entier i, j

Solution 2 (Tri par insertion – *7 points*)

1. **Spécifications :**
   La fonction `insert` x l *comp* ajoute l'élément x à sa place dans la liste l triée selon l'ordre donné par la fonction de comparaison *comp*.

```
# let rec insert x list comp =
    match list with
      | [] -> x::[]
      | e::l when comp x e -> x::e::l
      | e::l -> e :: insert x l comp ;;
val insert : 'a -> 'a list -> ('a -> 'a -> bool) -> 'a list = <fun>
```

2. **Spécifications :**
   La fonction `insertion_sort` *comp list* trie la liste *list* selon l'ordre donné par la fonction *comp*.

```
# let rec insertion_sort comp = function
      [] -> []
    | e::l -> insert e (insertion_sort comp l) comp ;;
val insertion_sort : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
```

Version terminale :

```
# let insertion_sort_term comp l =
    let rec sort accu = function
        [] -> accu
      | e::l -> sort (insert e accu comp) l
    in
    sort [] l ;;
val insertion_sort_term : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
```

Solution 3 (Association – *5 points*)

Spécifications :
La fonction `assoc k list` retourne la valeur (value) associée à la clé (key) `k` dans `list`, une liste de couples (*key*, *value*) (avec $key > 0$) triée par clés (key) croissantes. Si `k` n'est pas valide ou si aucun couple n'a pour clé `k`, elle déclenche une exception.

```
# let rec assoc k list =
      if k <= 0 then
        invalid_arg "k not a natural"
      else
        let rec search = function
            []        -> failwith "not found"
          | (key, value)::l -> if key = k then
                                 value
                               else
                                 if k < key then
                                   failwith "not found"
                                 else
                                   search l
        in
        search list ;;

  val assoc : int -> (int * 'a) list -> 'a = <fun>
```

Solution 4 **(Mystery –** ***2 points*****)**

1. **Spécifications :**
   Donner les résultats des évaluations successives des phrases suivantes. (Hors warning éventuels)

```
# let mystery = function
      [] -> failwith "..."
      |e::f::l -> (let rec aux_mystery m1 m2 = function
                       [] -> m2
                       |e::l -> if e < m1 then aux_mystery e m1 l
                                else if e < m2 then aux_mystery m1 e l
                                else aux_mystery m1 m2 l
                   in if e < f then aux_mystery e f l else aux_mystery f e l);;
val mystery : ('a list) -> 'a = <fun>

  mystery [1;3;4;2];;
   - : int = 2

  mystery [3.5;8.2;9.5;4.0];;
   - : float = 4.0

  mystery ['a'];;
Une erreur (Exception : match_failure)
```

2. **Spécifications :**
   Que retourne la fonction `mystery` ?
   La fonction mystery retourne le deuxième plus petit élément de la liste s'il existe.