Contrôle continu LI 012
Devoir sur table n° 1
Ébauche de corrigé

1. (a) Ce programme trie un tableau d'entiers en recopiant une à une les valeurs directement à leur place dans un nouveau tableau. Pour ce faire, on utilise une variante de l'algorithme classique de recherche de minimum : une fois une valeur repérée (comme minimale), on la "marque" pour ne plus la considérer lors de la recherche suivante. Cette marque est réalisée au moyen d'un tableau de booléens.

   (b) **Nombre de transferts** exactement égal à n : chaque clé est copiée une fois.

   Nombre de comparaisons : dans la fonction `champion_non_marque`, on fait systématiquement n tours dans la boucle principale ; cette fonction est appelée n fois dans le programme principal. Le nombre de comparaison est donc exactement égal à n^2.

   (c) Amélioration(s). (1) On peut essayer de réduire le nombre de comparaisons : (a) en ne comparant que si `tmarque[i]` est faux, et (b) en commençant la recherche du champion après le "saut" des cases marquées. Dans le meilleur des cas (tableau déjà trié), cela réduit beaucoup le nombre de comparaisons. Mais on reste d'ordre quadratique. (2) On peut aussi se débarasser du tableau `tmarque`, en remplaçant les valeurs dans le tableau initial par une valeur particulière : on gagne en place et en simplicité, mais pas en nombre de comparaisons.

   Bien sûr, on peut aussi revenir à un des algorithmes classiques vus en cours.

2. On ne demandait pas d'**implémenter** les primitives, mais de les **utiliser**.

```
for (i=1 ; i <= longueur(liste) ; i++)
  x = element(liste, i) ;
  if (strlen(x) > 10) printf("%s\n", x) ;
```

3. Avantages : (1) temps constant pour les opérations de suppression et d'insertion ; (2) pas de borne à la taille des données. Inconvénients : (1) pas d'accès direct aux éléments : parcours nécessaire pour toutes les opérations, (2) occupation mémoire légèrement plus importante (au moins 1 pointeur par donnée), et nécessité de gérer la mémoire libre ; (3) système plus complexe à programmer.