EA4 – Éléments d'algorithmique
TD n° 4

Exercice 1 : déroulement des tris classiques

On rappelle la description suivante du tri par sélection :

```
def triSelection(L) :
  res = []
  while L != [] :
    m = minimum(L)
    L.remove(m)
    res.append(m)
  return res
```

ainsi que celle du tri par insertion (en place dans un tableau) :

```
def triInsertion(T) :
  for i in range(1, len(T)) :
    x = T[i]
    for j in range(i, 0, -1) :     # pour j de i à 1 par pas de -1
      if T[j - 1] > x :
        T[j] = T[j - 1]
      else :
        T[j] = x
        break
  return T
```

1. La fonction `triInsertion` ci-dessus présente une erreur. Proposez une correction.
2. Rappeler la complexité de ces deux algorithmes.
3. On considère le tableau `T` suivant : | 5 | 1 | 8 | 7 | 6 | 3 | 2 | 4 |

   Décrire le déroulement du tri par sélection et du tri par insertion sur le tableau `T`.

Exercice 2 : permutations

Dans cet exercice, on considère des tableaux d'entiers représentant des permutations : un tableau `T` de longueur n représente une permutation si et seulement s'il contient tous les entiers compris entre 1 et n (nécessairement exactement une fois chacun).
Décrire des algorithmes pour les problèmes suivants et en donner la complexité en temps et en espace :

1. vérifier qu'un tableau représente une permutation ;
2. trier une permutation ;
3. calculer la permutation inverse ;
4. calculer le produit (la composée) de deux permutations.

Exercice 3 : tri à bulles

Le tri à bulles est un algorithme de tri qui consiste à faire remonter progressivement les plus grands éléments d'un tableau. L'algorithme parcourt le tableau et compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet du tableau, l'algorithme recommence l'opération `len(T) 1` fois. .

```
def triBulles(T) :
  for i in range(len(T) - 1, 0, -1):
    for j in range(i) :
      if T[j] > T[j + 1] :
        T[j], T[j + 1] = T[j + 1], T[j]
```

1. Prouver sa correction et calculer sa complexité (évaluée en nombre de comparaisons d'éléments et d'a□ectations).
2. Montrer que si aucun échange n'est fait à l'étape *i*, l'algorithme peut être arrêté.
3. Montrer que si, à l'étape *i*, aucun échange n'est fait après le rang *j* alors on peut réduire la borne de la boucle principale (sur *i*).
4. Écrire une fonction `triBullesOptimise` qui implémente ces deux optimisations. Quelle est sa complexité ?
5. Les grands éléments qui se trouvent au début du tableau remontent rapidement (on dit que ce sont des $lièvres$) alors que les petits éléments à la fin ne descendent que d'une case à chaque étape (on dit que ce sont des $tortues$). Proposer une version modifiée de `triBulles` appelée `triShaker` où les tortues avancent aussi vite que les lièvres.
6. Prouver sa correction et calculer sa complexité.
7. Ces algorithmes semblent particulièrement bien adaptés aux machines où la mémoire vive est faible et les données sont stockées sur une bande (une cassette magnétique par exemple). Quelle est leur complexité évaluée en nombre de déplacements de la bande ? (compter `1` par case)
8. Et la complexité des autres algorithmes de tri que vous connaissez évaluée de la même façon ?