## Algorithmique et Programmation 1 – TD - TP 9

ALGORITHMES DE TRI

**CORRECTION**

### Exercice 1 - Calculs élémentaires de complexité : tri par insertion

Soit l'algorithme, exprimé en Python, de tri par insertion vu en cours

```
1  def tri_insertion(liste):
2      """List --> None
3      Tri la liste donnée en paramètre"""
4      for i in range(1, len(liste)):
5          elem = liste[i]
6          pos = i
7          while pos > 0 and liste[pos - 1] > elem :
8              liste[pos] = liste[pos-1]
9              pos = pos - 1
10         liste[pos] = elem
```

1. Dérouler cet algorithme sur la liste `[8, 10, 5, 7]`

`liste = [8, 10, 5, 7]`
`for` $i$ de 1 à $4-1=3$

- $i = 1$, `elem = 10` ; `pos = 1`
  - ◇ Boucle `while`
    - ⋆ `(pos = 1) > 0 and (liste[pos-1] = 8) > 10` : False - sortie
  - ◇ `liste[1] = 10`, `liste = [8, 10, 5, 7]`
- $i = 2$, `elem = 5` ; `pos = 2`
  - ◇ Boucle `while`
    - ⋆ `(pos = 2) > 0 and (liste[pos-1] = 10) > 5` : True
      - ∗ `liste[2] = 10`, `pos = 1`
    - ⋆ `(pos = 1) > 0 and (liste[pos-1] = 8) > 5` : True
      - ∗ `liste[1] = 8`, `pos = 0`
    - ⋆ `(pos = 0) >0` : False (attention : évaluation paresseuse.) Sortie
  - ◇ `liste[0] = 5`, `liste = [5, 8, 10, 7]`
- $i = 3$, `elem = 7` ; `pos = 3`
  - ◇ Boucle `while`
    - ⋆ `(pos = 3) > 0 and (liste[pos-1] = 10) > 7` : True
      - ∗ `liste[3] = 10`, `pos = 2`
    - ⋆ `(pos = 2) > 0 and (liste[pos-1] = 8) > 7` : True
      - ∗ `liste[2] = 8`, `pos = 1`
    - ⋆ `(pos = 1) > 0 and (liste[pos-1] = 8) > 10` : False - sortie
  - ◇ `liste[1] = 7`, `liste = [5, 7, 8, 10]`

2. Quelle est la mesure de complexité à utiliser pour évaluer la complexité de cet algorithme ?

- Comparaisons : ligne 7
- Affectations : lignes 5, 8 et 10

Le nombre de comparaisons et d'affectations dépendent de la liste à trier et de sa taille.

3. Calculer la complexité de cet algorithme dans le meilleur des cas

Meilleur cas : la liste est déjà triée.

- **Comparaisons** : si la liste est triée, `liste[i - 1] > liste[i]` sera toujours faux, et donc `liste[pos - 1] > elem` sera faux au premier tour de boucle. Il y aura donc $n-1$ comparaisons (un pour chaque passage dans la boucle for)
- **Affect6ations** : on ne rentre jamais dans la boucle `while`, donc $2(n-1)$ affectations ($n-1$ affectations ligne 5, $n-1$ affectations ligne 10, pour chaque passage dans la boucle `for`)

4. Calculer la complexité de cet algorithme dans le pire des cas

Pire cas : la liste est triée en ordre inverse.

- **Comparaisons** : si la liste est triée en ordre inverse, `elem` sera à chaque fois plus petit que tous les éléments triés jusque là. Donc `liste[pos - 1] > elem` sera faux pour `pos` de $i$ à 1. On a donc :
  - ◇ pour $i = 1$ : 1 comparaison
  - ◇ pour $i = 2$ : 2 comparaisons
  - ⋮
  - ◇ pour $i = n-1$ : $n-1$ comparaisons

  Soient $1 + 2 + \ldots + (n-1) = \frac{n(n-1)}{2}$ comparaisons.
- **Affectations** :
  - ◇ Pour les affectations lignes 5 et 10, toujours $2(n-1)$ affectations.
  - ◇ Ligne 8 : on effectue une affectation à chaque tour de la boucle `while`. Il y en a donc autant que de comparaisons en pire cas : $\frac{n(n-1)}{2}$

**Exercice 2 - Calculs élémentaires de complexité : algorithme du drapeau**

On rappelle ci-dessous l'algorithme du drapeau à 3 couleurs vu en cours :

```
1  def drapeau(liste):
2      """List --> None
3      Tri la liste, contenant 3 couleurs R, J, B, donnée en paramètre"""
4      i = 0
5      j = 0
6      r = len(liste) - 1
7      while i <= r:
8          if liste[i] == "J":
9              i = i + 1
10         elif liste[i] == "B" :
11             echange(liste, j, i)
12             i = i + 1
13             j = j + 1
14         else :
15             echange(liste, r, i)
16             r = r - 1
```

1. Appliquer cet algorithme à l'instance `["J","R","R","B","J","J","J","R","B"]`

`liste = ["J","R","R","B","J","J","J","R","B"]`, $n = 9$
$i = 0, j = 0, r = 8$
`while` $i <= r$

- `liste[0] = "J"`, $i = 1$ ($j = 0, r = 8$)
- `liste[1] = "R"`,
  - ◇ `echange(liste, 8, 1)`, `liste = ["J","B","R","B","J","J","J","R","R"]`
  - ◇ $i = 1, j = 0, r = 7$
- `liste[1] = "B"`,
  - ◇ `echange(liste, 0, 1)`, `liste = ["B","J","R","B","J","J","J","R","R"]`
  - ◇ $i = 2, j = 1, r = 7$
- `liste[2] = "R"`,
  - ◇ `echange(liste, 7, 2)`, `liste = ["B","J","R","B","J","J","J","R","R"]`
  - ◇ $i = 2, j = 1, r = 6$
- `liste[2] = "R"`,
  - ◇ `echange(liste, 6, 2)`, `liste = ["B","J","J","B","J","J","R","R","R"]`
  - ◇ $i = 2, j = 1, r = 5$
- `liste[2] = "J"`, $i = 3, j = 1, r = 5$
- `liste[3] = "B"`,
  - ◇ `echange(liste, 1, 3)`, `liste = ["B","B","J","J","J","J","R","R","R"]`
  - ◇ $i = 4, j = 2, r = 5$
- `liste[4] = "J"`, $i = 5, j = 2, r = 5$
- `liste[5] = "J"`, $i = 6, j = 2, r = 5$

$i > r$, sortie de boucle, retour de `liste = ["B","B","J","J","J","J","R","R","R"]`

2. Calculer la complexité en nombre de comparaisons et d'échanges dans le pire et le meilleur cas, en fonction de $n_B$, $n_J$ et $n_R$ désignant respectivement le nombre d'éléments bleus, jaunes et rouges. On a $n_B + n_J + n_R = n$.

Le nombre de comparaisons et d'échanges ne dépend pas de la disposition des couleurs dans la liste, mais du nombre de chacune de ces couleurs .

- **Comparaisons** : Il y en a toujours $2n_B + n_J + 2n_R$. (à chaque fois qu'on a un bleu ou un rouge on fait 2 comparaisons, 1 pour les jaunes).
- **Echanges** : il y a un échange pour chaque rouge, un échange pour chaque bleu, et pas d'échange pour les jaunes. $n_B + n_R$ échanges.

**Exercice 3 - Tri de listes**

1. Implémenter, dans un même fichier, les algorithmes vus en cours de **tri par sélection**, **tri par insertion** et **tri par comptage**.
2. Générer une liste de 100 nombres aléatoires compris entre 0 et 1000.
3. Grâce à la méthode `time()` du module prédéfini `time`[1], calculez le temps pour faire trier les listes avec chacun de ces algorithmes.
   **Attention !** Les algorithmes de tri par insertion et tri par sélection modifient la liste donnée en entrée ! Vous devez donc créer des copies de la liste originelle pour lancer les tris sur la même liste. Afin de copier **le contenu d'une liste et non pas sa référence**, vous pouvez utiliser le constructeur `list()` : par exemple `liste_inser = list(liste_aleatoire)`. N'oubliez pas de faire des tests pour vérifier que vous lancez bien vos algorithmes sur les bonnes listes.
4. Calculez le temps d'exécution de la méthode `sort` du module prédéfini `numpy`[2].
   **Note :** La fonction `sort` utilise un algorithme plus complexe que ceux vus en cours : l'algorithme *quicksort*. Vous étudierez cet algorithme ultérieurement.
5. Comparez les performances de ces 4 algorithmes. Un exemple d'exécution est le suivant :

```
liste originelle: [875, 449, 319, 764, 594, 256, 117, 698, 778, 875, 431, 421, 204,
384, 290, 944, 577, 834, 639, 73, 278, 719, 290, 215, 290, 799, 127, 360, 506, 368,
124, 805, 571, 707, 75, 27, 426, 219, 637, 536, 269, 694, 617, 269, 935, 490, 481,
795, 144, 226, 680, 476, 502, 210, 344, 661, 303, 23, 795, 866, 227, 420, 471, 53,
310, 198, 713, 467, 753, 698, 437, 430, 148, 887, 79, 815, 559, 946, 792, 904, 187,
700, 143, 844, 210, 995, 897, 792, 183, 840, 568, 345, 252, 434, 842, 8, 303, 376,
852, 824]

liste triée : [8  23  27  53  73  75  79 117 124 127 143 144 148 183 187 198 204 210
210 215 219 226 227 252 256 269 269 278 290 290 290 303 303 310 319 344
345 360 368 376 384 420 421 426 430 431 434 437 449 467 471 476 481 490
502 506 536 559 568 571 577 594 617 637 639 661 680 694 698 698 700 707
713 719 753 764 778 792 792 795 795 799 805 815 824 834 840 842 844 852
866 875 875 887 897 904 935 944 946 995]

Durée du tri par insertion : 0.0003199577331542969
Durée du tri par selection : 0.0002949237823486328
Durée du tri par comptage : 0.0005359649658203125
Durée du tri par la méthode sort : 3.910064697265625e-05

Sur cet exemple, le tri par la méthode sort est plus rapide que le
tri par sélection,qui est plus rapide que le tri par insertion,
lui même plus rapide que le tri par comptage
```

6. Dupliquez les tests, en variant la longueur de la liste.

```python
import random
import time
import numpy

########################################
#    Génération d'une liste aléatoire    #
########################################

liste_aleatoire = []
TAILLE = 100
for i in range(TAILLE):
```

1. https://docs.python.org/fr/3/library/time.html#module-time
2. https://docs.scipy.org/doc/numpy/reference/generated/numpy.sort.html

```
    nb = random.randrange(1001)
    liste_aleatoire.append(nb)


############################
#    Tri par insertion     #
############################

def tri_insertion(liste):
    """List --> None
    Tri la liste donnée en paramètre"""

    for i in range(1, len(liste)):
        elem = liste[i]
        pos = i
        while pos > 0 and liste[pos - 1] > elem :
            liste[pos] = liste[pos-1]
            pos = pos - 1
        liste[pos] = elem


############################
#    Tri par sélection     #
############################

def echange(liste, i, j):
    """List x Int x Int --> None
    Echange les élements de liste en position i et j"""

    elem = liste[i]
    liste[i] = liste[j]
    liste[j] = elem

def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    longueur = len(liste)
    for i in range(longueur - 1):
        min = i
        for j in range(i, longueur):
            if liste[j] < liste[min]:
                min = j
        if min != i:
            echange(liste, i, min)

############################
#    Tri par comptage      #
############################

def tri_comptage(liste):
    """List --> List
    Tri la liste donnée en paramètre"""
    ind = []
    result = []
    n = len(liste)
    for i in range(n) : # initialisation de la liste d'indices et du tableau résultat final
        ind.append(0)
        result.append(0)
    for i in range(n - 1): #comptage
        for j in range(i+1, n) :
            if liste[j] > liste[i] :
                ind[j] = ind[j] + 1
```

```
            else :
                ind[i] = ind[i] + 1

    for i in range(n) : #tableau trié final
        result[ind[i]] = liste[i]
    return result

######################
#   Tests des temps   #
######################

liste_inser = list(liste_aleatoire)
#print("Liste avant le tri par insertion :", liste_inser)
debut_inser = time.time()
tri_insertion(liste_inser)
fin_inser = time.time()
temps_inser = fin_inser - debut_inser
#print("Liste après le tri par insertion :", liste_inser)

liste_select = list(liste_aleatoire)
#print("Liste avant le tri par sélection :", liste_select)
debut_select = time.time()
tri_selection(liste_select)
fin_select = time.time()
temps_select = fin_select - debut_select
#print("Liste après le tri par selection :", liste_select)

liste_comptage = list(liste_aleatoire)
#print("Liste avant le tri par comptage :", liste_comptage)
debut_comptage = time.time()
liste_comptage = tri_comptage(liste_comptage)
fin_comptage = time.time()
temps_comptage = fin_comptage - debut_comptage
#print("Liste après le tri par comptage :", liste_select)

liste_sort = list(liste_aleatoire)
#print("Liste avant le tri par la méthode sort :", liste_sort)
debut_sort = time.time()
liste_sort = numpy.sort(liste_sort)
fin_sort = time.time()
temps_sort = fin_sort - debut_sort
#print("Liste après le tri par la méthode sort :", liste_sort)

print("liste originelle:", liste_aleatoire)
print("")
print("liste triée :", liste_sort)
print("")

print("Durée du tri par insertion :", temps_inser)
print("Durée du tri par selection :", temps_select)
print("Durée du tri par comptage :", temps_comptage)
print("Durée du tri par la méthode sort :", temps_sort)

###################################
# Comparaison des temps d'exécution #
###################################

liste_temps = [temps_inser, temps_select, temps_comptage, temps_sort]
liste_temps = numpy.sort(liste_temps)

liste_algo = [0,0,0,0]
```

```
for i in range(4) :
    if liste_temps[i] == temps_inser :
        liste_algo[i] = "tri par insertion"
    elif liste_temps[i] == temps_select :
        liste_algo[i] = "tri par sélection"
    elif liste_temps[i] == temps_comptage :
        liste_algo[i] = "tri par comptage"
    else:
        liste_algo[i] = "tri par la méthode sort"

print("")
print("Sur cet exemple, le ", liste_algo[0], " est plus rapide que le ", sep="")
print(liste_algo[1], ",qui est plus rapide que le ", liste_algo[2], ",", sep="")
print("lui même plus rapide que le ", liste_algo[3], sep="")
```