LYCÉE VAUGELAS 2021/2022	
MPSI	Informatique
Corrigé du TD Semaine 11	29 – 30/11/2021

Quelques remarques générales

- L'algorithme étudié est très simple, l'objectif du TD est de réinvestir toutes les notions rencontrées dans les TD précédents.
- On prend garde à ne pas dégrader la complexité quadratique par un mauvais usage du langage (découpage des listes en particulier).
- On rappelle qu'une fonction qui travaille en place sur un tableau ne retourne rien (en réalité elle retourne `None` par défaut d'instruction `return`).

Exercice 1

Il faut commencer par représenter les schémas sur lesquels on s'appuie pour écrire les algorithmes, avec les indices utiles et leurs intervalles de variation.

1. Fonction `permute` qui permute les éléments d'indices `i` et `j` du tableau `t` :

```
def permute(t, i, j):
    t[i], t[j] = t[j], t[i]
```

2. Fonction `minimum(t,j)` qui retourne l'indice du minimum de la partie du tableau `t` comprise entre les indices `j` et `n-1` :

```
def minimum(t, j):
    pos, mini = j, t[j]
    for k in range(j + 1, len(t)):
        v = t[k]
        if v < mini:
            pos, mini = k, v
    return pos
```

3. On rappelle la définition du cours : dans une boucle `for`, l'invariant de boucle est une propriété qui est vraie à la fin du premier parcours de la boucle, puis qui reste vraie à la fin des parcours suivants du corps de la boucle.
   Voici un invariant pour la boucle de la fonction `minimum` :

$$t[\text{pos}] = \text{mini} \quad ; \quad \text{mini} = \min\{t[\ell], \ell \in [\![j, k]\!]\} \tag{1}$$

   La première partie est évidente puisque les deux variables sont modifiées simultanément. La seconde partie se montre facilement par récurrence sur k, puisque le corps de la boucle consiste justement à mettre à jour le minimum, ce qui assure l'hérédité. En sortie de boucle, on a donc

$$t[\text{pos}] = \text{mini} \quad ; \quad \text{mini} = \min\{t[\ell], \ell \in [\![j, n-1]\!]\} \tag{2}$$

   ce qui montre la correction de la fonction.

4. Algorithme de tri par sélection :

Tri par sélection impératif

```
Entrée : un tableau t à trier en place
Sortie : rien
fonction tri_selection(t)
    n <- taille(t)
    Pour j de 0 à n - 2
        i <- minimum(t, j)
        Si i <> j
            permute (t, i, j)
        Finsi
    FinPour
```

5. Fonction `tri_selection (t)` qui trie le tableau `t` **en place** par la méthode du tri par sélection :

```
def tri_selection(t):
    for j in range(len(t) - 1):
        i = minimum(t, j)
        if i != j:
            permute(t, i, j)
```

6. De nouveau on donne un invariant de fin de boucle `for` :

$$t[0:j+1] \text{ est trié} \qquad ; \qquad \forall k \in [\![j, n-1]\!],\ t[k] \geqslant t[j] \tag{3}$$

La démonstration se fait par récurrence sur j. L'initialisation est triviale avec $j = 0$ puisque le tableau ne contient qu'un élément inférieur ou égal à tous les autres. L'hérédité est aussi immédiate : on pioche un élément dans $[\![j, n-1]\!]$, supérieur à tous les éléments déjà en place et inférieurs à tous ceux qui restent à trier. La correction est ainsi démontrée puisque l'invariant fournit en sortie de boucle :

$$t[0:n] \text{ est trié} \tag{4}$$

7. Le nombre de comparaisons est identique dans le meilleur des cas, le cas moyen ou le pire des cas car les fonctions reposent sur des boucles `for` :

$$C(n) = \sum_{j=0}^{n-2} \sum_{k=j+1}^{n-1} 1 = \frac{n(n-1)}{2} \tag{5}$$

8. Les échanges ne sont pas systématiques, mais conditionnés par le succès du test de la fonction `tri_selection`.

Le meilleur des cas correspond à un tableau déjà trié : il n'y a aucun échange. Le pire des cas requiert $n-1$ échanges (tous les éléments sauf le dernier doivent être permutés). En ce sens le tri par sélection est optimal, avec un nombre minimal d'écritures en mémoire.

9. Les deux questions précédentes montrent que la complexité temporelle est dominée par les opérations réalisées dans la boucle de la fonction `minimum`. Le nombre de tests est représentatif, les autres opérations étant en nombre borné. La complexité temporelle est donc en $\Theta(n^2)$, ce qui est la caractéristique d'un tri quadratique.

10. La complexité spatiale est en $\Theta(1)$ car aucune structure auxiliaire n'est créée, à part un nombre borné de variables auxiliaires.

11. Le tri a été écrit pour qu'il soit stable : il faut pour cela utiliser une inégalité stricte dans la comparaison de la fonction minimum, pour ne pas permuter des éléments équivalents pour le préordre.

12. Le tri par sélection est un tri quadratique peu performant, notamment à cause de la complexité temporelle dans le meilleur des cas, ou dans le cas pratique de tableaux presque triés. Si l'on cherche à limiter les écritures en mémoire sur des dispositifs critiques, il peut être intéressant pour des tableaux de petite taille du fait de la propriété soulignée à la question 8 et précisée en fin d'énoncé.

13. Le test de la fonction `tri_selection` permet de ne pas réaliser systématiquement l'échange si l'élément d'indice j est déjà en place, ce qui est assez rare. Pour l'itération j, il reste à trier le tableau pour les indices allant de j à $n-1$, ce tableau a pour taille $n-j$. Le tri est une permutation de ce tableau, ou ce qui revient au même, une permutation de $[\![1, n-j]\!]$. Si l'on considère ces permutations comme équiprobables, l'échange se produit si 1 n'est pas un point fixe de la permutation. Or il y a $(n-j)!$ permutations dont $(n-j-1)!$ laissent le premier élément inchangé. La probabilité de ne pas réaliser d'échange est donc $\dfrac{(n-j-1)!}{(n-j)!} = \dfrac{1}{n-j}$. Le nombre moyen d'échanges à l'itération j est donc $\frac{n-j-1}{n-j}$. Le nombre d'échanges moyen vérifie donc la relation

$$E(n) = \sum_{j=0}^{n-2} \frac{n-j-1}{n-j} = n - 1 - \sum_{j=0}^{n-2} \frac{1}{n-j} = n - \sum_{i=1}^{n} \frac{1}{i} = n - H_n \tag{6}$$

où la n-ième somme partielle de la série harmonique est donnée par la formule d'Euler :

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = \ln n + \gamma + o(1) \tag{7}$$

On a donc une estimation du nombre d'échanges dans le cas moyen donné par

$$E(n) \sim n - \ln n - \gamma = \Theta(n) \tag{8}$$

Le cas moyen est identique au pire des cas en ordre de grandeur asymptotique. Il ne serait donc pas dramatique de supprimer le test de la fonction `tri_selection`.

14. La recherche récursive du minimum est davantage un exercice de style qu'une approche naturelle si l'on travaille sur un tableau. On privilégie la récursivité terminale, ce qui donne la fonction suivante.

```
def minimum(t, j, pos = -1, mini = 0, b = True):
    n = len(t)
    if j == n:
        return pos
    if  t[j] < mini or b:
        pos, mini = j, t[j]
    return minimum (t, j + 1, pos, mini, False)
```

La fonction `tri_selection` s'écrit simplement et naturellement sous la forme d'une fonction récursive terminale :

```
def tri_selection(t, j = 0):
    if j < len(t):
        i = minimum(t, j)
        if i != j:
            permute(t, i, j)
        tri_selection(t, j + 1)
```

15. La fonction `minimum` a une complexité temporelle qui vérifie (en simplifiant le cas $j = n$)

$$T_1(j) = \Theta(1) + T_1(j+1) \qquad ; \qquad T_1(n) = 0 \tag{9}$$

La récurrence fournit donc $T_1(j) = (n-j)\,\Theta(1)$.
La fonction `tri_selection` a une complexité qui s'écrit

$$T_2(j) = \Theta(1) + T_1(j) + T_2(j+1) \qquad ; \qquad T_2(n) = 0 \tag{10}$$

Par suite $T_2(j) = (n-j+1)\,\Theta(1) + T_2(j+1)$ et

$$T_2(0) = \sum_{k=0}^{n-1} (n-k+1)\,\Theta(1) = \left(\sum_{i=2}^{n+1} i\right)\Theta(1) = \Theta(n^2) \tag{11}$$

La complexité temporelle quadratique est respectée par la version récursive de l'algorithme. La complexité spatiale est en $\Theta(1)$ car la récursivité est terminale.

16. Pour tester, on développe autour de l'idée suivante :

```
from random import randint
T = [randint(0, 10) for i in range(10)]
print(T)
tri_selection(T)
print(T)
```

Fin