# EA4 – Éléments d'algorithmique
# TD n° 7 : tri et sélection rapide
## (Correction)

**Exercice 1 : déroulement des algorithmes de tris et sélection rapide**

On considère le tableau `T_1` suivant :

| 7 | 1 | 5 | 6 | 2 | 3 | 10 | 8 | 0 | 9 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|

Décrire le déroulement des variantes suivantes du tri rapide sur le tableau `T_1` :

**1.** variante avec mémoire auxiliaire, en prenant le premier élément comme pivot ;

**2.** *idem*, mais en choisissant toujours le pivot de manière optimale.

**3.** variante « en place », le pivot étant toujours le premier élément.

On considère maintenant le tableau `T_2` suivant :

| 73 | 19 | 55 | 64 | 28 | 37 | 99 | 82 | 5 | 91 | 46 |
|---|---|---|---|---|---|---|---|---|---|---|

**4.** Décrire le déroulement de l'algorithme de sélection rapide pour trouver l'élément de rang 4.

***Correction :***

**1.**

```
T = [7, 1, 5, 6, 2, 3, 10, 8, 0, 9, 4]
pivot = 7
T = [1, 5, 6, 2, 3, 0, 4]
pivot = 1
T = [0]
[0]
T = [5, 6, 2, 3, 4]
pivot = 5
T = [2, 3, 4]
pivot = 2
T = []
[]
T = [3, 4]
pivot = 3
T = [4]
[4]
[3, 4]
[2, 3, 4]
T = [6]
[6]
[2, 3, 4, 5, 6]
[0, 1, 2, 3, 4, 5, 6]
T = [10, 8, 9]
pivot = 10
T = [8, 9]
pivot = 8
T = []
[]
T = [9]
[9]
[8, 9]
T = []
[]
[8, 9, 10]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

**2.**

```
T = [7, 1, 5, 6, 2, 3, 10, 8, 0, 9, 4]
pivot = 5
T = [1, 2, 3, 0, 4]
pivot = 2
T = [1, 0]
pivot = 0
T = []
[]
T = [1]
[1]
[0, 1]
T = [3, 4]
pivot = 3
T = []
[]
T = [4]
[4]
[3, 4]
[0, 1, 2, 3, 4]
T = [7, 6, 10, 8, 9]
pivot = 8
T = [7, 6]
pivot = 6
T = []
[]
T = [7]
[7]
[6, 7]
T = [10, 9]
pivot = 9
T = []
[]
T = [10]
[10]
[9, 10]
[6, 7, 8, 9, 10]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

***3.*** *En place, l'ordre des éléments change beaucoup plus. On supposera que le pivot est gardé en première position jusqu'à la fin du partitionnement. Les partitions gauches et droites sont agrandies jusqu'à ce que deux éléments doivent être échangés et ainsi de suite. Ensuite le pivot est échangé avec le dernier élément de la partition gauche.*

*On détaille le premier partitionnement :*

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *Début* | *7* | *[]1* | *5* | *6* | *2* | *3* | *10* | *8* | *0* | *9* | *4[]* |
| *Agrandissement partitions* | *7* | *[1* | *5* | *6* | *2* | *3]* | *10* | *8* | *0* | *9* | *4[]* |
| *Échange 10 et 4* | *7* | *[1* | *5* | *6* | *2* | *3* | *4]* | *8* | *0* | *9* | *[10]* |
| *Agrandissement partitions* | *7* | *[1* | *5* | *6* | *2* | *3* | *4]* | *8* | *0* | *[9* | *10]* |
| *Échange 8 et 0* | *7* | *[1* | *5* | *6* | *2* | *3* | *4* | *0]* | *[8* | *9* | *10]* |
| *Échange pivot et 0* | *[0* | *1* | *5* | *6* | *2* | *3* | *4]* | ***7*** | *[8* | *9* | *10]* |
| *Ensuite :* | *[[]**0*** | *[1* | *5* | *6* | *2* | *3* | *4]]* | *7* | *[8* | *9* | *10]* |
| *À gauche, rien à faire* | *[0* | *[1* | *5* | *6* | *2* | *3* | *4]]* | *7* | *[8* | *9* | *10]* |
| *Etc* | *[0* | *[[]**1*** | *[5* | *6* | *2* | *3* | *4]]]* | *7* | *[8* | *9* | *10]* |
| | *[0* | *[1* | *[5* | *6* | *2* | *3* | *4]]]* | *7* | *[8* | *9* | *10]* |
| | *[0* | *[1* | *[[3* | *4* | *2]* | ***5*** | *[6]]]]* | *7* | *[8* | *9* | *10]* |
| | *[0* | *[1* | *[[[2]* | ***3*** | *[4]]* | *5* | *[6]]]]* | *7* | *[8* | *9* | *10]* |
| | *[0* | *[1* | *[[**2*** | *3* | ***4***] | *5* | *[6]]]]* | *7* | *[8* | *9* | *10]* |
| | *[0* | *[1* | *[2* | *3* | *4* | *5* | ***6***]]] | *7* | *[8* | *9* | *10]* |
| | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *[[]**8*** | *[9* | *10]]* |
| | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *[8* | *[[]**9*** | *[10]]* |
| | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *[8* | *[9* | ***10***]] |
| | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |

***4.*** *On effectue les partitionnements en place, en choisissant toujours le premier élément comme pivot et en échangeant à la fin le pivot avec le dernier élément de la plage des éléments plus petits.*

*Apres le premier partitionnement (avec pivot* 73*), le tableau devient :*

$$[5, 19, 55, 64, 28, 37, 46, 73, 82, 91, 99]$$

*Le pivot a abouti en position* 8*. Puisque position* $= 8 > k = 4$ *on fait un appel récursif* `selection_rapide(gauche, k)`*, soit* `selection_rapide([5, 19, 55, 64, 28, 37 46], 4)`*.*

*Cet appel effectue le second partitionnement (pivot* 5*), le tableau ne change pas (cas le pire, le pivot est le minimum, aucun élément à gauche du pivot, tous les autres éléments à droite) :*

$$[5, 19, 55, 64, 28, 37, 46]$$

*Le pivot a abouti en position* 1*. Puisque position* $= 1 < k = 4$ *on fait un appel récursif* `selection_rapide(droite, k - position)`*, soit* `selection_rapide([19, 55, 64, 28, 37 46], 3)`*.*

*Cet appel effectue un partitionnement avec* 19 *(le minimum, encore) comme pivot. Le tableau ne change pas non plus :*

$$[19, 55, 64, 28, 37, 46]$$

*Le pivot a abouti en position* 1*. Puisque position* $= 1 < k = 3$ *on fait un appel récursif* `selection_rapide(droite, k - position)`*, soit* `selection_rapide([55, 64, 28, 37, 46], 2)`*.*

*Cet appel effectue un partitionnement avec pivot* 55*. Le tableau devient :*

$$[37, 46, 28, 55, 64]$$

*Le pivot a abouti en position* 4*. Puisque position* $= 4 > k = 2$ *on fait un appel récursif* `selection_rapide(gauche, k)`*, soit* `selection_rapide([37, 46, 28], 2)`*.*
*Cet appel effectue un partitionnement avec pivot* 37*. Le tableau devient :*

$$[28, 37, 46]$$

*Le pivot a abouti en position* 2*. Puisque position* $= 2 = k$ *on retourne la valeur du pivot :* 37*.*

### Exercice 2 : hauteur de pile du tri rapide

Comme tout algorithme récursif, le tri rapide est sujet à des débordements de la pile si le nombre d'appels récursifs devient trop grand.

**1.** On considère tout d'abord l'implémentation du tri rapide avec copies de tableaux suivante :

```
def triRapide(T) :
    if len(T) <= 1 : return T
    pivot = T[0]
    gauche = [elt for elt in T[1:] if elt <= pivot]
    droite = [elt for elt in T[1:] if elt > pivot]
    return triRapide(gauche) + [pivot] + triRapide(droite)
```

Quelle hauteur atteint la pile dans le pire cas ? dans le meilleur cas ?

**2.** On peut dérécursiver le deuxième appel récursif (terminal). Quelle hauteur atteint alors la pile sur l'entrée $[1, \ldots, n]$ ? Et sur l'entrée $[n, \ldots, 1]$ ?

**3.** Proposer une solution permettant de garantir une hauteur de pile maximale en $O(\log n)$ dans tous les cas (où $n$ est la longueur du tableau à trier).

**4.** Quel est alors le meilleur cas en ce qui concerne la hauteur de pile ?

***Correction :***

***1.*** *La hauteur de la pile sera égale à la hauteur de l'arbre des appels récursifs. Dans le pire cas, à chaque appel récursif l'un des sous-tableau est vide et l'autre contient tous les éléments sauf le pivot, d'où une hauteur de pile égale à* $n$*.*
*Dans le meilleur cas, on divise par deux la taille du tableau à chaque appel récursif donc la hauteur de pile est un* $\Theta(\log n)$*.*

***2.*** *Sur l'entrée* $[1, \ldots, n]$*, l'algorithme choisit* 1 *(le minimum) comme pivot. La partition* `gauche` *sera alors vide.*
*Seulement l'appel récursif* `triRapide(gauche)` *est effectué, c'est-à-dire* `triRapide([])`*, cela correspond à un empilement. Cet appel exécute seulement le cas de base (il retourne* `[]`*) et il s'arrête sans faire d'autres appels récursifs. Il n'y aura pas d'autres empilements. La pile atteint la hauteur* 1 *(ou* 2 *si on compte aussi l'empilement relatif à l'appel initial* `triRapide([1,..., n])`*).*

*Sur l'entrée* $[n, \ldots, 1]$*, l'algorithme choisit* $n$ *(le maximum) comme pivot. La partition* `gauche` *est alors* $[n-1, \ldots, 1]$*. Seulement l'appel récursif* `triRapide(gauche)` *est effectué c'est-à-dire*

`triRapide([n-1, ..., 1])`, *cela correspond à un empilement.*

*Cet appel partitionne le tableau avec* $n-1$ *comme pivot, la partition* `gauche` *sera* $[n-2,\dots,1]$. *Seulement l'appel récursif* `triRapide(gauche)` *est effectué, c'est-à -dire* `triRapide([n-2, ..., 1])`, *cela correspond à un autre empilement, et ainsi de suite.*

*Pour tout* $i = n, n-1, \dots, 1$ *il y aura un appel récursif et donc un empilement. La pile atteint la hauteur* $n$ *(ou* $n+1$ *si on compte aussi l'empilement relatif au premier appel à* `triRapide([n,..., 1])`*).*

3. *On détermine d'abord quelle est la plus petite parmi les deux partitions* `gauche` *et* `droite` *et on fait en premier l'appel récursif sur la plus petite parmi les deux, et ensuite le second l'appel récursif sur la plus grande.*
   *Le second appel est alors terminal et peut être dérécursivé sans utiliser d'espace sur la pile (à optimiser manuellement suivant les langages, cela requiert un tri en place).*
   *Le premier des deux appels (le seul effectué) sera toujours fait sur un tableau de taille strictement plus petite que la moitié de la taille de* $T$. *Le nombre total d'appels est donc inférieur à* $\log_2 n$.
4. *Le meilleur cas en terme de hauteur de pile est lorsqu'aucun appel récursif n'a besoin d'être effectué, ce qui arrive lorsqu'un des deux sous-tableaux est vide, ce qui correspond au pire cas (en temps de calcul) du tri rapide.*

## Exercice 3 : des vis et des écrous

Un charpentier dispose d'un tas d'écrous et de vis et souhaite associer chaque écrou à la vis correspondante. Chaque écrou correspond à une seule et unique vis (et réciproquement). En essayant de comparer une vis à un écrou, le charpentier peut voir lequel est le plus grand, mais ne peut pas comparer directement deux vis ou deux écrous. Comment peut-il alors associer efficacement chaque vis à son écrou ?

*Indication : penser au tri rapide !*

▷ *Adaptation du tri rapide : on choisit un écrou* pivot. *On partitionne l'ensemble des vis par rapport à ce pivot, ce qui permet d'identifier la vis* pivot *correspondant à l'écrou pivot et de séparer les vis en deux groupes : celles plus petites et celles plus grandes que la vis pivot. La vis pivot peut être utilisée de la même façon pour partitionner les écrous. Etc.*

## Exercice 4 : tri drapeau

Le problème du *drapeau hollandais* est le suivant : le tableau à trier contient trois types d'éléments, les bleus, les blancs et les rouges et on souhaite les trier par couleur. Tous les éléments d'une même couleur ne sont pas identiques, il ne suffit donc pas de les compter. On dit qu'un algorithme de tri est *stable* s'il préserve l'ordre du tableau initial pour les éléments de même couleur.

1. Proposer un algorithme linéaire et stable (mais pas en place) pour résoudre ce problème.

   ▷

```
BLEU, BLANC, ROUGE = 0, 1, 2
def drapeau(T) :
```

```
    bleus,blancs,rouges=[],[],[]
    for couleur,contenu in T:
        if couleur == BLEU:
            bleus.append((couleur,contenu))
        elif couleur == BLANC:
            blancs.append((couleur,contenu))
        else:
            rouges.append((couleur,contenu))
    return bleus+blancs+rouges
```

**2.** En vous inspirant de l'algorithme de partition en présenté en cours, proposer un algorithme linéaire et en place (mais pas stable) dans le cas où il n'y a que deux couleurs.

▷ *Réponse 1 :*
*On maintiendra l'invariant suivant :*

> « le tableau est constitué de 3 segments consécutifs, un bleu, un non exploré et un rouge ».

*Le curseur i représente la frontière entre le segment bleu et le segment non exploré. Le curseur j représente la frontière entre le segment non exploré et le segment rouge.*

```
def drapeau(T) :
    i=0
    j= len(T)-1
    while i <= j :
        if T[i] == BLEU :
            i += 1
        elif T[j] == ROUGE :
            j -= 1
        else :
            T[i], T[j] = T[j], T[i]
    return T
```

*Réponse 2 :*
*On maintiendra l'invariant suivant :*

> « le tableau est constitué de 3 segments consécutifs, un bleu, un rouge et un non exploré ».

*Le curseur i représente la frontière entre le segment bleu et le segment rouge. Le curseur j représente la frontière entre le segment rouge et le segment non exploré.*

```
def drapeau(T) :
    i=0
    j= 0
    while j <= len(T)-1:
        if T[i] == BLEU :
            T[i], T[j] = T[j], T[i]
            i += 1
            j += 1
        else :
            j += 1
    return T
```

**3.** Généraliser cet algorithme au cas de trois couleurs. Pour cela, on maintiendra l'invariant suivant :

> *« le tableau est constitué de 4 segments consécutifs, un bleu, un blanc, un non exploré, et un rouge ».*

Utiliser trois curseurs pour représenter les frontières entre ces segments.

▷

```
def drapeau(T) :
  i = 0
  j = 0
  k = len(T)-1
  while j <= k :
    if T[j] == BLEU :
      T[i], T[j] = T[j], T[i]
      i += 1
      j += 1
    elif T[j] == BLANC :
      j += 1
    else :
      T[j], T[k] = T[k], T[j]
      k -= 1
  return T
```

**4.** En déduire un algorithme de tri rapide en place et optimisé dans le cas d'un pivot apparaissant plusieurs fois dans le tableau.

▷

```
def triRapide(T) :
  return triRapideAux(T, 0, len(T)-1)

def triRapideAux(T, deb, fin) :
  if fin-deb+1 == 1 : return [T[deb]]
  if fin-deb+1 < 1 : return []
  pivot = T[deb]
  i = deb
  k = fin
  j = i
  while j <= k :
     if T[j] < pivot :
       T[i], T[j] = T[j], T[i]
       i += 1
       j += 1
     elif T[j] == pivot :
       j += 1
     else :
       T[j], T[k] = T[k], T[j]
       k -= 1
  return triRapideAux(T, deb, i-1) + [pivot] * (k-i+1) + triRapideAux(T, j, fin)

triRapide(T)
```

**5.** Faire tourner l'algorithme précédent sur le tableau suivant :

| 7 | 1 | 9 | 6 | 7 | 3 | 10 | 7 | 0 | 1 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|

▷

| 7 | 1 | 9 | 6 | 7 | 3 | 10 | 7 | 0 | 1 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $ij$ | | | | | | | | | | $k$ |

, $T[j] = pivot$, *avance* $j$.

| 7 | 1 | 9 | 6 | 7 | 3 | 10 | 7 | 0 | 1 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | $j$ | | | | | | | | | $k$ |

, $T[j] < pivot$, *échange* $T[i]$ *et* $T[j]$, *avance* $i$ *et* $j$.

| 1 | 7 | 9 | 6 | 7 | 3 | 10 | 7 | 0 | 1 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | $i$ | $j$ | | | | | | | | $k$ |

, $T[j] > pivot$, *échange* $T[j]$ *et* $T[k]$ *(deux* 9*), recule* $k$ .

| 1 | 7 | 9 | 6 | 7 | 3 | 10 | 7 | 0 | 1 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | $i$ | $j$ | | | | | | | $k$ | |

, $T[j] > pivot$, *échange* $T[j]$ *et* $T[k]$, *recule* $k$ .

| 1 | 7 | 1 | 6 | 7 | 3 | 10 | 7 | 0 | 9 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | $i$ | $j$ | | | | | | $k$ | | |

, $T[j] < pivot$, *échange* $T[i]$ *et* $T[j]$, *avance* $i$ *et* $j$.

| 1 | 1 | 7 | 6 | 7 | 3 | 10 | 7 | 0 | 9 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $i$ | $j$ | | | | | $k$ | | |

, $T[j] < pivot$, *échange* $T[i]$ *et* $T[j]$, *avance* $i$ *et* $j$.

| 1 | 1 | 6 | 7 | 7 | 3 | 10 | 7 | 0 | 9 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | $i$ | $j$ | | | | $k$ | | |

, $T[j] = pivot$, *avance* $j$.

| 1 | 1 | 6 | 7 | 7 | 3 | 10 | 7 | 0 | 9 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | $i$ | | $j$ | | | $k$ | | |

, $T[j] < pivot$, *échange* $T[i]$ *et* $T[j]$, *avance* $i$ *et* $j$.

| 1 | 1 | 6 | 3 | 7 | 7 | 10 | 7 | 0 | 9 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | $i$ | | $j$ | | $k$ | | |

, $T[j] > pivot$, *échange* $T[j]$ *et* $T[k]$, *recule* $k$ .

| 1 | 1 | 6 | 3 | 7 | 7 | 0 | 7 | 10 | 9 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | $i$ | | $j$ | $k$ | | | |

, $T[j] < pivot$, *échange* $T[i]$ *et* $T[j]$, *avance* $i$ *et* $j$.

| 1 | 1 | 6 | 3 | 0 | 7 | 7 | 7 | 10 | 9 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | $i$ | | $jk$ | | | |

, $T[j] = pivot$, *avance* $j$.

| 1 | 1 | 6 | 3 | 0 | 7 | 7 | 7 | 10 | 9 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | $i$ | | $k$ | $j$ | | |

, $j$ *a dépassé* $k$, *partitionnement complété.*

*On recommencera récursivement le même procédé sur* $[1, 1, 6, 3, 0]$ *et* $[10, 9, 9]$.

*Remarque : il n'existe pas à notre connaissance d'algorithme à la fois linéaire, stable et en place pour ce problème ; on peut en revanche modifier légèrement l'algorithme de la question 3 pour obtenir un algorithme stable et en place (mais pas linéaire, donc).*