UFR IM²AG
UNIVERSITÉ Grenoble Alpes

**L3 Informatique**
**Année 2017–2018**

# UE ALGO5 — TD2 — Séance 3 : Tri par segmentation

On rappelle le schéma algorithmique du tri par segmentation :

```
TriSegmentation(T):
    si la taille de T est > 1 alors
        Soit p une valeur «pivot»
        Partitionner T en T1, T2 t.q.:
        – T1 contient les valeurs de T inférieures ou égales à p
        – T2 contient les valeurs de T supérieures à p
        TriSegmentation(T1)
        TriSegmentation(T2)
```

## Exercice 1. Réalisation

---

**Corrigé —**

Avant la première question on peut réfléchir à la signature de TriSegmentation : arriver assez vite à la conclusion que «passer un tableau» en paramètre d'une procédure récursive n'est pas forcément raisonnable (recopies de tableaux). On travaille sur des tranches disjointes : on peut donc implémenter un «tableau» par un tableau (global) + deux indices (indice de début, indice de fin). Il n'y a ainsi pas de copie de tableaux inutiles.

```
T : un tableau sur [1..N] d'entiers

TriSegmentation(données p, q : deux entiers)
{ Trie le sous−tableau T[p..q].
  Pré-condition : 1 ≤ p ≤ q ≤ N
  Post-condition : T[p..q] trié }

Appel initial : TriSegmentation(1,N)
```

---

*Q 1.* Le partitionnement sera réalisé par une procédure intermédiaire nommée Partition. Donner la spécification de cette procédure (signature, pré- et post-conditions).

---

**Corrigé —**

L'idée est ici de réfléchir à la spécification sans réaliser tout de suite la procédure.
—Comme pour TriSegmentation, il s'agit de partitionner un sous-tableau : deux données pour l'indice de début et de fin.
—Valeur du pivot ? Est-elle déterminée à l'intérieur de Partition ou connue à l'appel de procédure ? Pas beaucoup d'importance : on suppose ici que c'est à l'intérieur (pas besoin de cette valeur après Partition)
—Comment sont déterminés $T_1$ et $T_2$ pour les appels subséquents à TriSegmentation ? ⇒ nouvel indice intermédiaire (dernier indice de $T_1$, premier de $T_2$... peu importe mais à définir).
Par exemple :

```
Partition(données p, q : deux entiers; résultat r : un entier)
{ Pré-condition : 1 ≤ p ≤ q ≤ N. Soit T0 la valeur initiale de T.
  Post-condition : p ≤ r ≤ q
  Il existe x une valeur de T0[p..q] telle que :
    – T[p..r] contient toutes les valeurs de T0 inférieures ou égales à x
    – T[r+1..q] contient toutes les valeurs de T0 supérieures à x
    – T[r] = x
}
```

*Q 2.* Donner une réalisation de TriSegmentation en utilisant Partition.

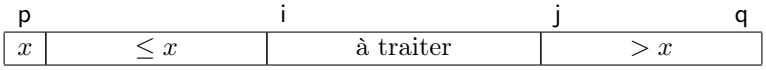
**Corrigé** —

```
TriSegmentation(p,q):
    r : un entier
debut
    si p < q alors
        Partition(p,q,r)
        TriSegmentation(p,r−1)
        TriSegmentation(r+1,q)
    finsi
fin
```

*Q 3.* Donner une réalisation de Partition : commencer par exprimer sous forme de schéma l'invariant de la boucle principale.

**Corrigé** —

Exemple d'invariant :




Algorithme associé :

```
Partition(p,q,r):
    i,j : deux entiers
    x : un entier
debut
    x <− T[p]
    i <− p + 1
    j <− q + 1
    tant que i < j
        si T[i] <= x alors
            i <− i + 1
        sinon
            j <− j − 1
            échanger(T[i],T[j])
        fin si
    fin tant que
    r <− i − 1
    échanger(T[p],T[r])
fin
```

## Exercice 2. Analyse

*Q 1.* Quel est le coût dans le pire cas, dans le meilleur cas de Partition ? de TriSegmentation ? Donner des exemples (de taille 10) de pire et meilleurs cas.

**Corrigé** —

Partition : coût de $O(n)$ dans tous les cas ($n$=taille du sous-tableau ; $n$ passages dans la boucle puisque $i$ incrémenté ou $j$ décrémenté à chaque itération).

TriSegmentation :
—Meilleur cas : à chaque appel récursif, partition en deux sous-tableau de taille identique.
Coût $C(n) = 2.C(n/2) + O(n) = O(n \log n)$
Exemple (à construire à partir des appels récursifs) :

```
                Tri(1,10)
          [5,2,1,4,3,9,8,7,6,10]
               /         \
       Tri(1,4)          Tri(6,10)
       [3,2,1,4]         [8,7,6,10,9]
       /     \            /     \
  Tri(1,2)  Tri(4,4)  Tri(6,7)  Tri(9,10)
    [1,2]     [4]       [6,7]     [9,10]
```

—Pire cas : à chaque appel récursif, partition en deux sous-tableau de taille $(n-1)$ et 0.
Coût $C(n) = C(n-1) + O(n) = O(n^2)$
Exemple : tous les éléments du tableau ont la même valeur.

---

*Q 2.* Donner le coût de Partition(T) dans le cas où tous les éléments de $T$ ont la même valeur. Proposer une correction de Partition susceptible d'améliorer ce coût.

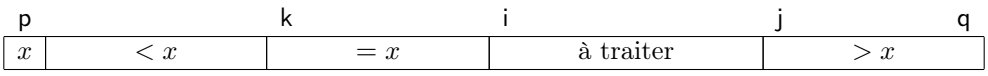
---

**Corrigé —**

On peut remarquer que la question se pose de manière plus générale pour les tableaux comportant un nombre important de fois la même valeur (ce qui arrive dans de nombreuses applications).

Identifier le problème : la partition effectuée met toutes les valeurs "égales" au pivot d'un seul côté. Il faudrait au contraire minimiser la taille de la plus grande partition.

Plusieurs pistes possibles :
—les valeurs égales au pivot peuvent très bien être placées dans l'une ou l'autre partition : alterner entre les deux (à l'aide d'un booléen par exemple) ?
—plus rigoureux : faire un vrai «drapeau hollandais», en regroupant toutes les valeurs égales au milieu du tableau.




```
Partition(p,q,r):
  i,j,k : trois entiers
  x : un entier
debut
        x <- T[p]
        i <- p + 1
        k <- p + 1
        j <- q + 1
        tant que i < j
                si T[i] = x alors
                        i <- i + 1
                sinon si T[i] < x alors
                        échanger(T[i],T[k])
                        i <- i + 1
                        k <- k + 1
                sinon
                        j <- j - 1
                        échanger(T[i],T[j])
                fin si
        fin tant que
        échanger(T[p],T[k-1])
        r <- max(k-1,min((p+q)/2,i-1))
fin
```

On peut ensuite remarquer que le sous-tableau contenant les valeurs égales au pivot est trié : on peut changer

la spécification de Partition pour donner comme résultat les trois partitions (on ajoute un résultat s, T[r..s] est le sous-tableau contenant les valeurs égales au pivot). L'algorithme de tri devient :

```
1 TriSegmentation(p,q):
      r : un entier
3 debut
      si p < q alors
5         Partition(p,q,r,s)
          TriSegmentation(p,r−1)
7         TriSegmentation(s+1,q)
      fin si
9 fin
```

---