# *Corrigé série 2 : Complexité des algorithmes*

**Exercice 1 :**

Il est bien connu que pour échanger les valeurs de deux variables, il faut une troisième variable intermédiaire.

a. Écrire un algorithme qui échange les valeurs de **a** et **b** à l'aide d'une troisième variable.

Cependant, pour les données numériques entières, il existe un moyen de ne pas recourir à une variable supplémentaire.

b. Écrire un algorithme qui échange les valeurs de **a** et **b** sans utiliser une autre variable. Est-ce que le coût de l'opération diminue ?

Solution

a. Algorithme permuter1 ;

```
Var a, b, x : entier ;
Début
  x←a ;  a←b ; b←x ;
Fin ;
```

b. Algorithme permuter2 ;

```
Var a, b, x : entier ;
Début
  a←a+b ;  b←a-b ; a←a-b ;
Fin ;
```

Le coût n'a pas diminué car :
permuter1 coût = 3 opérations d'affectations
permuter2 coût = 3 affectations+1 addition + 2 soustractions = 6 opérations.
Globalement les deux sont en O(1).

***Exercice 2:*** *Complexité en fonction de deux paramètres*

Déterminer la complexité des algorithmes suivants (par rapport au nombre d'itérations effectuées), où $m$ et n sont deux entiers positifs.

***Algorithme A***

$i \leftarrow 1$ ; $j \leftarrow 1$
tant que ($i \leq m$) *et* ($j \leq n$) faire
    $i \leftarrow i+1$
    $j \leftarrow j+1$
fin tant que
O(min(m,n))

***Algorithme B***

$i \leftarrow 1$ ; $j \leftarrow 1$
tant que ($i \leq m$) *ou* ($j \leq n$) faire
    $i \leftarrow i+1$
    $j \leftarrow j+1$
fin tant que
O(max(m,n))

***Algorithme C***

$i \leftarrow 1$ ; $j \leftarrow 1$
tant que ($j \leq n$) faire
  si $i \leq m$
  alors
    $i \leftarrow i+1$
sinon
    $j \leftarrow j+1$
fin si
fin tant que
O(m+n)

***Algorithme D***

$i \leftarrow 1$ ; $j \leftarrow 1$
tant que ($j \leq n$) faire
  si $i \leq m$
  alors
    $i \leftarrow i+1$
sinon
    $j \leftarrow j+1$ ;  $i \leftarrow 1$
fin si
fin tant que
O(mxn)

**Exercice 3:**

Algorithme 1

```
r ← 0
for ( i ← 1 ; i ≤ n² ; i←i+1)
        for ( j ← 1 ; j ≤ 2n – 1 ; j←j+1)
                        r ← r + 1
```

$$\sum_{i=1}^{n^2}\sum_{j=1}^{2n-1} 1 = \sum_{i=1}^{n^2}(2n-1) = n^2(2n-1) = 2n^3 - n^2 \Rightarrow \Theta(n^3)$$

Algorithme 2

```
r←0 ; i← 1 ;
tantque (i ≤ n) faire
     pour j← n² à  5 avec pas -1 faire
            r←r+1;
     finpour;
 i←i+2 ;
fintq ;
retourner r ;
```

$i=1$ $(n^2 – 4)$ itérations
$i=2$ «
$i=4$ «
$i=8$ «
… …
$i=n$ $(n^2 – 4)$ itérations
On a donc au pire cas : $n/2^* (n^2 – 4)=1/2(n^3 – 4n) \sim O(n^3)$

Algorithme 3

```
r ← 0
for (i ← 1 ; i ≤ n ; i ← i +1)
     for (j ← i +1 ; j ≤ n ; j ← j +1)
            for (k ← 1 ; k ≤ j ; k ← k + 1)
                      r ← r + 1
```

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\sum_{k=1}^{j}1=\sum_{i=1}^{n}\sum_{j=i+1}^{n}j=\sum_{i=1}^{n}\sum_{j=1}^{n}j-\sum_{j=1}^{i}j$$

$$=\sum_{i=1}^{n}\left(\frac{n(n+1)}{2}-\frac{i(i+1)}{2}\right)$$

$$=\frac{1}{2}\sum_{i=1}^{n}(n^2+n)$$

$$-\frac{1}{2}\sum_{i=1}^{n}i^2$$

$$-\frac{1}{2}\sum_{i=1}^{n}i=\frac{1}{2}n(n^2+n)-\frac{1}{2}\frac{n(n+1)(2n+1)}{6}-\frac{1}{2}x\frac{n(n+1)}{2}$$

$$=\frac{1}{2}n^3+\frac{1}{2}n^2-n^2-n=O(n^3)$$

***Exercice 4 :***

Écrire un algorithme qui calcule, pour un tableau T de taille n donné, la somme :

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(T(i)-T(j))^2$$

a. Donner un algorithme « naïf » de complexité quadratique.

**b.** Donner un algorithme de complexité linéaire.

S←0 ;
Pour i←1 à n faire
    Pour j←1 à n faire   S←S+(T[i] – T[j])*(T[i] – T[j]) ;   fait ;
fait ; complexité $O(n^2)$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(T(i)-T(j))^2=\sum_{i=1}^{n}\sum_{j=1}^{n}T(i)^2+\sum_{i=1}^{n}\sum_{j=1}^{n}T(j)^2-2\sum_{i=1}^{n}\sum_{j=1}^{n}T(i)T(j)$$

$$=n*\textstyle\sum_{i=1}^{n}T(i)^2+n*\sum_{j=1}^{n}T(j)^2-2*\sum_{i=1}^{n}T(i)*\sum_{j=1}^{n}T(j)$$

$$=2n\textstyle\sum_{i=1}^{n}T(i)^2-2(\sum_{i=1}^{n}T(i))^2$$

Donc $S=2nS_2-2(S1)^2$ où $S_2=\sum_{i=1}^{n}T(i)^2$ et $S_1=\sum_{i=1}^{n}T(i)$

D'où l'algorithme de complexité linéaire est le suivant
S2←0 ; S1←0 ;
Pour i←1 à n faire  S2←S2+T[i]*T[i] ;   S1←S1+T[i] ; fait ;
S←2*n*S2–2*S1*S1 ;
Retourner S ;

**Exercice 5 :** Soit la fonction suivante :

```
Fonction M(n : entier) : entier ;    //spécification {n>0}
Var i, S : entier ;
Début
   S ← 0 ;
   pour i ← 1 jusqu'à n faire  S ← S + (2 * i*i);   fait ;
  retourner(S);
Fin;
```

a. Que fait cette fonction?
   Réponse : elle calcule

$$2\sum_{i=1}^{n} i^2$$

b. Donner sa complexité.
   Réponse : boucle pour de 1 à n donc la complexité est de l'ordre de O(n)

c. Pourrait-on faire mieux pour obtenir le même résultat ? Si oui donnez son coût.
   Réponse : oui on peut l'améliorer comme suit :
   Fonction M(n : entier) : entier ;
   Début
      return $\frac{n(n+1)(2n+1)}{3}$ ;
   Fin;
   Complexité de l'ordre de O(1)

***Exercice 6 :*** *Interclassement de deux tableaux triés*

On dispose de deux tableaux *T1[1..n]* et *T2[1..n]* dont les éléments sont triés de façon croissante. On veut créer un tableau trié *T3[1..2n]* contenant tous les éléments de *T1* et *T2*. Pour cela on propose deux algorithmes Fusion_A et Fusion_B.

*Fusion_A* : initialise T3 avec T1 (déjà trié) et y insère un à un les éléments de T2 de façon à ce que l'ordre soit respecté.

*Fusion_B* : remplit T3 en parcourant simultanément T1 et T2 du début jusqu'à leur fin. Soit i1 et i2 les indices courant dans T1 et T2, on a 3 cas possible :

*Si T1[i1]<T2[i2] alors mettre T1[i1] à la fin de T3 et avancer dans T1*
*Si T1[i1]>T2[i2] alors mettre T2[i2] à la fin de T3 et avancer dans T2*
*Sinon mettre T1[i1] puis T2[i2] à la fin de T3 et avancer dans T1 et T2*

1. Ecrire les deux algorithmes et déroulez sur l'exemple :

T1= | 1 | 3 | 5 | et T2 = | 2 | 3 | 4 |

2. Donnez la complexité, au pire des cas, des algorithmes en fonction de la taille des données.
3. Quel algorithme choisissez-vous d'implémenter ?

**Corrigé :**

```
Action  Fusion_A(E/ t1, t2 :tableau[1..n] d'entiers ;E/n :entier ;
                                        S/ t3 :tableau[1..2n]d'entiers)
i, j, k, m :entier ;
Debut
Pour i :=1 à n faire t3[i]=t1[i] ; finpour ;
m:=n ;
Pour j :=1 à n faire
   i :=1 ;
   tantque (i<=m et t3[i]<t2[j]) i :=i+1 ; finTq//  recherche de la position
   k :=m ;
   tantque (k>=i) faire t3[k+1] :=t3[k] ; k :=k-1 ; finTq ; // décalage à droite
   t3[i] :=t2[j] ; m :=m+1 ;
Finpour ;
Fin ;
```

Complexité de Fusion_A est de l'ordre de $O(n^2)$

```
Action  Fusion_B(E/ t1, t2 :tableau[1..n] d'entiers ;E/n :entier ;
                                        S/ t3 :tableau[1..2n]d'entiers)
i, j, k :entier ;
Debut
   i :=1; j :=1; k :=1 ;
   tantque (i<=n et j<=n) faire
      si (t1[i]<t2[j]) alors t3[k] :=t1[i]; i :=i+1;
      sinon si (t1[i]>t2[j]) alors t3[k] :=t2[j] ; j :=j+1 ;            O(n) ou
            sinon t3[k] :=t1[i]; k:=k+1; t3[k]:=t2[j]; i:=i+1; j:=j+1 ;  O(2n-1)
            finsi ;
      finsi ;
    k :=k+1 ;
   finTq;
   tantque (i<=n) faire t3[k] :=t1[i] ; k :=k+1; i :=i+1 ; finTq ;
   tantque (j<=n) faire t3[k] :=t2[j] ; k :=k+1; j :=j+1 ; finTq ;  O(1+n) ou
                                                                     O(2)
Finpour ;
Fin ;
```

Complexité de Fusion_B est de l'ordre de $O(2n+1) \approx O(n)$
On choisira l'algorithme Fusion_B car il a une complexité linéaire.