Série 2 : Complexité des algorithmes

<u>Exercice 1:</u> *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

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

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

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

<u>Exercice2 :</u>

Déterminer un algorithme qui teste si un tableau de taille n est un "tableau de permutation" (i.e. tous les éléments sont distincts et compris entre 1 et n).

1. Donner un premier algorithme naïf qui soit quadratique.
2. Donner un second algorithme linéaire utilisant un tableau auxiliaire.
3. Donner un troisième algorithme linéaire sans utiliser un tableau auxiliaire.

<u>Exercice 3 :</u> *Produit matriciel*

On considère deux matrices carrées (d'entiers) d'ordre n, A et B. Le produit de A par B est une matrice carrée C définie par :

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} * B_{k,j}$$

1. Donner un algorithme calculant le produit de deux matrices représentées sous forme d'un tableau à deux dimensions. Calculer la complexité de cet algorithme.
   Doit-on préciser dans quels cas (pire cas, meilleur des cas, cas moyen) cette complexité est obtenue ?
2. Modifier l'algorithme précédent lorsque la matrice A est de dimension (m,n) et la matrice B de dimension (n, p). Quelle est alors la complexité de l'algorithme ?

<u>Exercice 4</u> : Tri d'un tableau ne contenant que des 0 et des 1

On souhaite trier un tableau T de n entiers appartenant à l'ensemble {0,1} de façon à ce que les valeurs nulles soient rangées au début du tableau. Par exemple:

T=									
	0	1	1	0	0	…	1	1	1

Après l'application de l'algorithme de tri on aura :

T=									
	0	0	0	0	1	…	1	1	1

Au cours du traitement, une partie des données est déjà triée et le tableau est organisé de la façon suivante :

	1			i			j				n
T=	0	0	0	?	?	?	?	1	1	1	1

Les ? représentent les données non encore traitées.

1. Que représentent i et j ?
2. Selon la valeur de T[i] quel traitement doit-on effectuer ?
3. Quand doit-on arrêter l'algorithme ?
4. Écrire l'algorithme de tri et donner sa complexité en temps.

<u>Exercice 5 :</u> *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 ?

***Exercice 6* : Recherche séquentielle**

On considère un tableau *A* de *n* éléments, que l'on suppose trié en ordre croissant.
On cherche à construire un algorithme permettant de savoir à quel endroit du tableau se trouve une valeur *clé*. (On suppose que *clé* est bien dans le tableau et on recherchera la première occurrence de cette valeur.)

1. Donner un algorithme itératif qui résout ce problème. Indiquer et démontrer un invariant de boucle pour cet algorithme.
2. A quoi correspond le pire des cas ? En déduire la complexité en *O* de l'algorithme.
3. A quoi correspond le meilleur des cas ? En déduire la complexité en *O* de l'algorithme.
4. Ecrire cet algorithme sous forme récursive et l'exécuter sur le tableau suivant avec *clé*=18.

1	7	8	9	12	15	18	22	30	31

5. Prouver l'algorithme récursif.
6. Comment faut-il modifier ces versions (itérative et récursive) de l'algorithme si l'on n'est pas sûr que *clé* appartienne au tableau ?

***Exercice 7* : Recherche dichotomique**

On se place dans les mêmes conditions que celles de l'exercice précédent (*A* est un tableau trié de *n* éléments).

1. Donner un algorithme itératif qui recherche une clé par la méthode dichotomique.
2. Développer cet algorithme sur le tableau suivant avec *clé* = 30.

1	7	8	9	12	15	18	22	30	31

3. Indiquer et démontrer un invariant de boucle pour cet algorithme.
4. On suppose que le tableau *A[1..n]* contient $n=2^k$ éléments (où K est un entier positif). Combien d'itérations l'algorithme effectuera-t-il au maximum ?
5. En déduire la complexité (en *O*) de l'algorithme.
6. Pour k=100 comparer les complexités des algorithmes de recherche séquentielle et dichotomique.
7. Ecrire l'algorithme sous forme récursive et l'exécuter sur l'exemple de la question 1.
8. Déterminer la complexité (en *O*) de l'algorithme récursif.
9. Comment faut il modifier ces versions (itérative et récursive) de l'algorithme si l'on n'est pas sûr que *clé* appartienne au tableau ?

***__Exercice 8__* : Tri sélection**

Le tri sélection d'un tableau T[1..n] de n éléments consiste, pour i variant de 1 à n-1, à déterminer l'élément minimum du sous-tableau T[i..n] et à échanger cet élément avec T[i].

a. On note T[d..f] le sous-tableau de T compris entre les indices d et f. Ecrire une fonction itérative *rech_min(T, d, f)* qui retourne l'indice du plus petit élément de T[d..f]. Prouver la terminaison et la validité de cette fonction.
b. Déterminer la complexité de la fonction *rech_min*.
c. On considère la procédure suivante :

```
Procédure  Tri_Sélection(T : tableau ; n :entier)
{  i, k : entier ;
     pour i :=1 à n-1 faire
          k=rech_min(T, i, n) ;
          si (i ≠ k) alors échanger (T[i], T[k])  finsi;
   fait ;  }
```

Déterminer la complexité de la procédure *Tri_sélection*.