Exercice 1

Exprimer les fonctions suivantes en notation O , Ω puis θ :

$f_1(n)=3n$, $f_2(n)=2^n$ $f_3(n)=n^2$, $f_5(n)=n^n$,
$f_6(n)=\log n$, $f_7(n)=n!$, $f_8(n)= n\log n$

Exercice 2

Pour chacun des fonctions $T_i(n)$ suivant, déterminer sa complexité asymptotique dans la notation Grand-O.

$T_0(n) = 3n$
$T_1(n) = 6n^3 + 10n^2 + 5n + 2$
$T_2(n) = 3\log_2 n + 4$
$T_3(n) = 2^n + 6n^2 + 7n$
$T_4(n) = 7k + 2$
$T_4(n) = 4\log_2 n + n$
$T_5(n) = 2\log_{10} k + kn^2$

Exercice 3

Considérer les deux algorithmes A1 et A2 avec leurs temps d'exécution $T_1(n) = 9n^2$ et $T_2(n) = 100n + 96$ respectives.

1. Déterminer la complexité asymptotique des deux algorithmes dans la notation Grand-O.
   Quel algorithme a la meilleure complexité asymptotique ?
2. Montrer que vos solutions sont correctes en spécifiant un c et un n_0 par algorithme afin que la relation suivante soit satisfaite : $O(f) = \{g|\exists c > 0 : \exists n_0 > 0 : \forall n \geq n_0 : g(n) \leq cf(n)\}$
3. Calculer les temps maximales d'exécution des deux algorithmes $T_i(n)$ pour $n = 1, n = 3, n = 5, n = 10, n = 14$.
4. Ebaucher les graphes des deux fonctions T_i dans un même système de coordonnés (abscisse n, ordonné $T_i(n)$).
5. Etudier quel algorithme est le plus efficace en fonction de n?
6. Quelle est la complexité asymptotique de l'algorithme suivant ? Quelle règle avez vous appliqué ?

```
début
    appeler A1
    appeler A2
fin
```

Exercice 4

Addition de matrices

Considérer les deux matrices quadratiques A et B de taille n :

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix},$$

L'addition de ces deux matrices donne la matrice C quadratique de taille n:

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix}$$

Avec $cij = aij + bij$ $\forall i, \forall j$

1. Définir le type des matrices quadratiques et déclarer les variables A,B, et C.
2. Ecrire un algorithme qui effectue l'addition des deux matrices A et B et stocke les résultats en C.
3. Déterminer la fonction de temps maximale ("worst case") T(n) pour des matrices de taille n.
4. Déterminer la complexité Grand-O pour des matrices de taille n.

Exercice 5

Multiplication de matrices

La multiplication des deux matrices quadratiques de taille n donne la matrice C quadratique de taille n:

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix}$$

avec

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad \forall i, \forall j$$

1. Ecrire un algorithme qui effectue la multiplication des deux matrices A et B et stocke les résultats en C.
2. Déterminer la fonction de temps maximale (”worst case”) T(n) pour des matrices de taille n.
3. Déterminer la complexité O(n) pour des matrices de taille *n*.

Exercice 6

Écrire l'algorithme qui recherche un élément dans un vecteur de taille n. Calculer la complexité temporelle en fonction du nombre de comparaisons dans le pire et dans le meilleur des cas. Refaire les calculs en fonction du nombre d'accès au vecteur.

Exercice 7

Écrire l'algorithme de tri par sélection. Calculer la complexité temporelle en fonction de nombre de comparaisons et de permutations dans le pire et dans le meilleur des cas. Refaire les calculs en fonction du nombre d'accès au vecteur.

Exercice 8

On considère deux manières de représenter ce que l’on appelle des « matrices creuses », c'est-à-dire des matrices d’entiers contenant environ 90% d’éléments nuls :

a) La matrice est représentée par un tableau à deux dimensions dont les cases contiennent les éléments.
b) La matrice est représentée par un tableau à une dimension. On ne s’intéresse qu’aux éléments de la matrice *qui ne sont pas nuls*. Chaque case du tableau contient un triplet (i, j, a) correspondant à l’indice de ligne, l’indice de colonne, et la valeur d’un élément non nul.

Le problème considéré consiste à calculer la somme des éléments d’une matrice. On demande d’écrire un algorithme permettant de calculer cette somme, pour chacune des deux représentations, puis de comparer leur complexité spatiale (espace mémoire occupé) et leur complexité temporelle (nombre d’opérations à effectuer). Que peut-on conclure de cette comparaison ? Montrer qu’il existe une valeur critique du nombre d’éléments non nuls à partir de laquelle une méthode l’emporte sur l’autre.

Exercice 9

Pour chacun des algorithmes suivants évaluer le nombre d'opérations :

Algo 1 affichage des n composantes du vecteur x
Pour i allant de 1 à n **faire**
    afficher(xi)

Algo 2
Pour i allant de 1 à n **faire**
  Pour j allant de 1 à n **faire**
    afficher(xi+xj)

Algo 3
Pour i allant de 1 à n **faire**
 Pour j allant de 1 à n **faire**
  Pour k allant de 1 à n **faire**
    Pour l allant de 1 à n **faire**
      Pour m allant de 1 à n **faire**
afficher(xi+xj+xk+xl+xm)

Exercice 10

On considère, pour effectuer la recherche d’un élément dans un tableau, la recherche séquentielle et la recherche dichotomique. On s’intéresse à leur complexité temporelle.

Pour cela, considérer un tableau ayant mille éléments (version trié, et version non trié). Pour chaque algorithme, et pour chaque version du tableau, combien de comparaisons sont à effectuer pour :

- trouver un élément qui y figure ?
- trouver un élément qui n'y figure pas ?

Quels sont les cas où le tableau est parcouru complètement et les cas où un parcours partiel est suffisant ?
Conclure en donnant la complexité temporelle pour chaque algorithme

Exercice 11

Soient les 3 algorithmes suivants permettant de calculer la valeur d'un polynôme
$p(x) = a0+a1x+a2x^2+...+anx^n$
en un point x. Comparer leurs complexités.

algo 1
début
    p = a[0] ;
    pour i := 1 **à** n **faire**
        calculer p+ = a[i]* x^i ;

algo 2
p = a[0] ;
q = 1;
pour i = 1 **à** n **faire**
    q = q * x ;
    p := p + a[i] * q ;

algo 3
p = a[n] ;
pour i := n **à** 1, **pas** -1, **faire**
    p := p*x + a[i-1] ;

Exercice 12

On considère un tableau à une dimension contenant des lettres majuscules. On désire compter la fréquence de chacune des 26 lettres de l'alphabet. Ecrire deux procédures qui donnent en sortie un tableau de fréquence: l'une où le tableau est parcouru 26 fois, et l'autre (plus performante !) où le calcul est fait en un seul parcours. On pourra supposer que l'on dispose d'une fonction auxiliaire *position(lettre)* qui pour chaque lettre donne sa position dans l'alphabet : *position('A') = 1, ..., position('Z) = 26.*

Exercice 13

Soit l'algorithme suivant qui effectue la recherche dichotomique du rang (place) d'un nombre A dans une suite triée (par ordre croissant) de n nombres mis dans un tableau à une dimension (vecteur L[i]; i=1,..,n). Cet algorithme fonctionne sous l'hypothèse que A est présent dans la liste.

Algorithme :
début
    place = 1 ;
    f = n ;
tant que place < f **faire** milieu = (place + f) / 2 ; **si** L[milieu] < A
alors place = milieu + 1 ;
    sinon f = milieu ;
    finsi; **fait**; **fin**;

Question Donner la complexité de cet algorithme.

Exercice 14

Écrire l'algorithme de tri à Bulles. Calculer la complexité temporelle en fonction de nombre de comparaisons et de permutations dans le pire et dans le meilleur des cas. Refaire les calculs en fonction du nombre d'accès au vecteur. Comparer avec le tri par sélection.

Exercice 15

Calculer la complexité temporelle de l'algorithme de recherche dichotomique en fonction du nombre de comparaisons dans le pire et dans le meilleur des cas. Refaire les calculs en fonction du nombre d'accès au vecteur. Comparer avec l'algorithme de recherche séquentiel.

Exercice 16

On considère trois tris élémentaires : le tri sélection, le tri par insertion (trois variantes), et le tri bulle. On considère pour un tableau les deux cas extrêmes où le tableau est déjà trié (dans l'ordre croissant), et celui où il est trié dans l'ordre décroissant. Décrire avec précision le comportement et la complexité de chacun des algorithmes dans ces deux cas. Quelles conséquences peut-on en tirer ?

Exercice 17

Compléter le tableau ci-dessous

Nombre d'opérations en fonction de la taille n des données	**Avec un ordinateur X**		**Avec un ordinateur Y 100 fois plus rapide que X**	
	Taille maximale (n max) des problèmes traités en 1h	Nombre d'opérations effectuées en 1h	Taille maximale (n max) des problèmes traités en 1h	Nombre d'opérations effectuées en 1h
5n				
Log n				
n^2				
2^n				