Corrigé type

Exo1 :

```
Algorithme Exo1 ;
Var M :entier ;
Fonction Fib(n : entier ) :entier ;
Debut
  si (n=0) ou (n=1) alors
        Fib← 1 ;
  Sinon
        Fib ← Fib(n-1) + Fib (n-2) ;
Fin ;
Debut
        Lire (M) ;
        Ecrire ('le nombre de fibonacci numero',M,'est ', Fib(M)) ;
fin.
```

Exo2

```
Fonction paire (n : entier) : booleen ;
Debut
        Si (n=0)alors
          Paire ← true ;
        Sinon
          Paire ← impaire (n-1) ;
Fin ;

Fonction impaire (n :entier) : booleen
Debut
        Si (n=0)alors
          impaire ← false;
        Sinon
          impaire ← paire (n-1) ;
Fin ;
```

Exo3 :

```
.....
Var T : array [1..50] : entier ;
.....
Fonction RDRec ( min : entier, max : entier, x :entier) : entier ;
Var milieu : entier
Debut
si (min > max) alors
        RDRec ← -1
Sinon
        Milieu ← (max +min) /2 ;
        Si ( T[milieu]= x) alors
                RDRec ← milieu ;
        Sinon
                Si (x< T[milieu]) alors
                        RDRec ← RDRec (min, milieu-1, x)
                Sinon
                        RDRec ← RDRec (milieu+1, max, x) ;
                Finsi ;
        Finsi ;
Finsi ;
```

Exo4 :

1) Le temps nécessaire au traitement des tailles du probleme : n=10, n=100, n =1000 pour une unité de temps égale a une milliseconde est montré ci-dessus :

Algorithme	Complexité temporelle	Temps		
		N= 10	N= 100	N= 1000
Alg0	$O(1)$	0,001s	0,001s	0,001s
Alg1	$O(\log_2 n)$	0,002s	0,005s	0,009s
Alg2	$O(n)$	0,01s	0,1s	1s
Alg3	$O(n^2)$	0,1s	10s	16 min 40s
Alg4	$O(n^3)$	1s	16 mn 40s	11j 13h 46min 40s
Alg5	$O(2^n)$	1,02s	$3.2 *10^{16}$ millénaire	$3.2 *10^{286}$ millénaire

2) Le temps nécessaire au traitement des tailles du problème : n=10, n=100, n =1000 pour une unité de temps égale a une microseconde est montré ci-dessus :

Algorithme	Complexité temporelle	Temps		
		N= 10	N= 100	N= 1000
Alg0	$O(1)$	10^{-6} s	10^{-6} s	10^{-6} s
Alg1	$O(\log_2 n)$	$2,3 * 10^{-6}$ s	$4,6 * 10^{-6}$ s	$9,9 * 10^{-6}$ s
Alg2	$O(n)$	10^{-5} s	10^{-4} s	10^{-3} s
Alg3	$O(n^2)$	10^{-4} s	10^{-2} s	1 s
Alg4	$O(n^3)$	10^{-3} s	1s	16 min 40s
Alg5	$O(2^n)$	10^{-3} s	$3.2 *10^{13}$ millénaire	$3.2 *10^{283}$ millénaire

3) nous concluons que l'augmentation de la performance de la machine de calcule apporte les effets suivants :

a. Améliore le temps de calcul pour des complexités polynomiales
b. N'atténue en rien les valeurs prohibitives des complexités exponentielles des grandes tailles de pbm et ne peut donc pas constituer une solution pour contourner le pbm de l'explosion combinatoire
c. Pour les petites tailles, la fonction exponentielle est plus intéressante que certaines fonction polynomiales (n=10 Alg5 est egale a Alg4) voir plus rapide si on prend un autre ALG de l'ordre de N^4

Exo 5

Pour calculer l'ordre :

C(N) par rapport C(n+1) ?

Rappel :
C(n+1) - c(n)= 0 ➔ O(1)
C(n+1) - c(n) =1, 2 ,3 ..... ➔ O(N)
C(n+1) - c(n) = n, 2n+1, ... ➔ $O(N^2)$
C(2n) – C(n)= 1, 2, 3 ..... ➔ O(log n)

```
Fonction f1 (n)
Debut
        total=0
        pour i allant de 1 jusqu'à  n-1  faire
                pour j allant de i+1 jusqu'à  n  faire
                        total=total+1
        f1= total;
Fin
```

Calculer le nombre d'opérations pour N :

- **L'instruction hors boucle :**
  total=0 ➔ 1 opérations
  f1== total ➔ 1 opérations

- **Les 2 boucles imbriqués:**

  - ✓ On commence toujours par calculer le nombre d'instructions dans la boucle intérieur :
    L'instruction : total= total +1 **contient 2 opérations de base (addition et affectation)**.

  - ✓ PBM : Cette instruction se répète X fois / X est variable puisque il dépend de « **i** » !!!
  - ✓ La boucle d'extérieur s'exécute (N-1 fois) de 1 ➔ n-1

N =10
I : 1➔ 9

I=1 : (j=2➔10) donc 9 ops
I=2 : (j=3➔10) donc 8 ops
I=3 : (j=4➔10) donc 7 ops
.
.
I=9 : (j= 10➔10) donc 1 ops

N=11
I : 1➔10

I=1 : (j=2➔ 11) donc 10 ops
I=2 9 ops
I=3 8 ops
.
.
I=9 2 ops
I=10 1 ops

C'est une suite numérique dont R =1 ➔ somme = (nombre de termes)* (U1 + Un)/2 .....**(1)**

N=10

U1= 1
Un = 9 ➔ Un= (10-1) = N-1

Nombre de termes = de 1 jusqu'à 9 ➔ 9 termes ➔ (10-1) = N-1
Selon (1) :

Somme = 9+8+7+6+5+..... +1 = (N-1)* (1 + N-1)/ 2 = (N-1)*N/2

Donc les 2 boucles se répète : N*(N-1)/2 fois.
Et pour chaque itération, on a 2 opérations de base (total = total +1).
➔ Dans la boucle, on a **2 * somme = 2 * (N*(n-1) /2) = N*(N-1)**

- **C(N) = ?**

C(N) = 2 + N*(N-1) = N^2-N+2

- **C(N+ 1)** = ?

C(N+1) **=** $(N+1)^2 - (N+1) + 2$ = N^2+N+2

- **C(N+1) – C(N) = ???**

C(N+1) – C(N) = (N^2+N+2) – (N^2-N+2) = 2N << N^2 ➔ O(N^2)

Ainsi : 2 boucles imbriquées. Chacune de l'ordre de n ➔ algo est de l'ordre de O (n^2)

Exo5 Factorielle (preuve par récurrence)

1) **L'algorithme 2 calcule-t-il bien une factorielle ?**

Ben oui, il calcule bien ce qu'il faut. ➔ On le prouve par récurrence

a. Pour n = 0 on a bien que Factorielle(0) = 1 = 1!.
b. On suppose donc que pour n est vrai

Factorielle(n) = n!.

c. Pour n+ 1. L'algorithme retourne (n +1) * factorielle(n).

Par hypothèse de récurrence Factorielle(n) = n!,

➔ donc Factorielle(n+1) =(n+l) * Factorielle(n) = (n+l)*n! = (n+1) !

2)**Quel est le domaine de définition l'entier N de l'algorithme 2 ? Que se passe-t-il si N est en dehors de son domaine ?**

Rep :

Le domaine de définition est l'ensemble des entiers naturels. Si N est un réel ou un entier négatif, l'algorithme ne s'arrête pas. Pour pallier ce problème on peut changer le test si N=0 alors rendre 1 par si N <= 0 alors rendre 1.

3)On n'a pas une boucle ! Comment peut on calculer la complexité dans ce cas ?

- **Rappel :**

  Si on a programme X contient 3 instructions, et un appel a une fonction F1

  Donc : C (PGM)= 3 + C(F1) // Attention : C(F1) n'est égale pas a 1. Il faut le calculer.

- Le problème principale se réside dans l'instruction qui fait un appel récursif : **fact = N * fact (N-1)**

  On note C' (Fact(n)) (C prime) : le nombre d'opérations de base pour cette instruction

  Donc : C'(fact (n)) = C'(instr) = C' (fact(n-1)) + 2 ops (2ops : une opération de multiplication + une opération d'affectation)

  On a : C'(fact(n)) ⇔ C'(n)

  ➔ C'(N) = C'(N-1)+2 ➔ C'(N)- C'(N-1)= 2 .....(1)

- Calculer la relation entre C(N) et C(N-1) de la fonction complète :

  Si (N=1) ➔ 1 opérations

  Fact = N* fact (N-1) ➔ C'(N) opération

  C(N) = 1 + C'(N)

  C(N-1) = 1 + C'(N-1)

  Donc : C(N)-C(N-1) = [1 + C'(N)] - [1+ C'(N-1)]

  C(N)-C(N-1) = C'(N) – C'(N-1) = 2 ..... { selon l'équation 1 }

  C(N+1)- C(N) = C(N)- C(N-1) = 2 << N ➔ O(N)

exo7 : l'addition de 2 polynômes

```
Donnees
        (n) : degré max des polynômes
        P, Q les deux tableaux correspondant aux polynômes
 Debut
        de i=0 a i=n faire
                R[i] = P[i]+Q[i]
Rendre R
Fin
```

- Pour calculer la complexité, on commence tjr par calculer C(n) et C(n+1)

  On a une seule boucle de 0 jusqu'à n , cad (N +1) itération.

  Pour chaque itération, on a une addition et une affectation : cad 2 instructions

  C(n) = 2N + 2 .

  C(n+1)= 2(n+1)+2 = 2N + 4

Donc la différence entre C(n) et C(n+1) =2 ➔ complexité de l'ordre de N ➔ O(n)

Exo8 : La multiplication des 2 polynômes :

```
Données
        (n) : degré max des polynômes
        P, Q les deux tableaux correspondant aux polynômes
Début
        de i=0 a i=2n faire .....(1)
                R[i] = 0 ......(i)

        de i=0 a i=n faire .....(2)
                de j=0 a j=n faire .....(3)
                        R[i+j] = R[i+j] + P[i]*Q[j] .....(ii)

Rendre R
Fin
```

- Complexité

Dans l'instruction (ii), on a : multiplication, 3 addition, affectation = 5 opérations de base

- Da La boucle (1) ➔ 2n+1 ops
- Dans la boucle intérieur (3) on a : N+1 itération ➔ 5*(N +1) opérations

  La boucle (3) et (2) sont imbriquées, on va exécuter la boucle (3) ... N+1 fois➔ (5*(N+1))*(N+1) = $5N^2$+ 10N + 5

Total C(n)= [2N+1] + [$5N^2$+10N+5] = $5N^2$ + 12N+6

- Calculer l'ordre de cet Algorithme :

C(n) = $5N^2$ + 12N+6

C(n+1) = 5 $(N+1)^2$+ 12 (N+1) +6 = $5N^2$+22N+28

C(n+1) – C(n) = 10N + 22 << N^2 ➔ de l'ordre de (N^2)